5-2008

Security hardened remote terminal units for SCADA networks.

Jeff Hieb
University of Louisville

SECURITY HARDENED REMOTE TERMINAL UNITS FOR SCADA NETWORKS

By

Jeffrey Lloyd Hieb
B.S., Furman University, 1992
B.A., Furman University, 1992
M.S., University of Louisville, 2004

A Dissertation
Submitted to the Faculty of the
Graduate School of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

Department of Computer Science and Computer Engineering
J. B. Speed School of Engineering
University of Louisville
Louisville, Kentucky

May 2008

SECURITY HARDENED REMOTE TERMINAL UNITS FOR SCADA NETWORKS

By

Jeffrey Lloyd Hieb
B.S., Furman University, 1992
B.A., Furman University, 1992
M.S., University of Louisville, 2004

A Dissertation Approved on

February 26, 2008

By the following Dissertation Committee members

Dr. James H. Graham, Dissertation Director

Dr. Adel Elmaghraby

Dr. Patricia A. S. Ralston

Dr. Dar-jen Chang

Dr. Rammohan K. Ragade

DEDICATION

To my wife

Jennifer

for all of her support, encouragement and patience.

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to Dr. James Graham, my dissertation director. Collaborating and working with him has been both pleasurable and educational. His technical guidance, perspective and encouragement were essential to the completion of this dissertation. I wish to thank the members of my committee, Dr. Patricia Ralston, Dr. Adel Elmaghraby, Dr. Dar-jen Chang, and Dr. Rammohan K. Ragade for their time, expertise, and support. A special thanks to Dr. Adel Elmaghraby for mentoring me in the future faculty program. I learned a lot from our conversations. I also owe special thanks to my friends and colleagues Sandy Patel, Rob Kelley, Doug Wampler, and Nathan Johnson for distracting me when I need distracting and assuring me when I need assuring. Many thanks to Dr. Walden Laukhuf, Mr. Steve Williamson, and the Chemical Engineering Department. Generous use of their facilities, particularly the Process Control Laboratory and the Unit Operations Laboratory, as well as their time and expertise were greatly appreciated. Thanks also to Ron Lile and his staff for providing technical support.

ABSTRACT

SECURITY HARDENED REMOTE TERMINAL UNITS FOR SCADA NETWORKS

Jeffrey L. Hieb

February 26, 2008

Remote terminal units (RTUs) are perimeter supervisory control and data acquisition (SCADA) devices that measure and control actual physical devices. Cyber security was largely ignored in SCADA for many years, and the cyber security issues that now face SCADA and DCS, specifically RTU security, are investigated in this research. This dissertation presents a new role based access control model designed specifically for RTUs and process control. The model is developed around the process control specific data element called a point, and point operations. The model includes: assignment constraints that limit the RTU operations that a specific role can be assigned and activation constraints that allow a security administrator to specify conditions when specific RTU roles or RTU permissions cannot be used.

RTU enforcement of the new access control model depends on, and is supported by, the protection provided by an RTU's operating system. This dissertation investigates two approaches for using minimal kernels to reduce potential vulnerabilities in RTU protection enforcement and create a security hardened RTU capable of supporting the new RTU access control model. The first approach is to reduce a commercial OS kernel

to only those components needed by the RTU, removing any known or unknown vulnerabilities contained in the eliminated code and significantly reducing the size of the kernel. The second approach proposes using a microkernel that supports partitioning as the basis for an RTU specific operating system which isolates network related RTU software, the RTU attack surface, from critical RTU operational software such as control algorithms and analog and digital input and output.

In experimental analysis of a prototype hardened RTU connected to real SCADA hardware, a reduction of over 50% was obtained in reducing a 2.4 Linux kernel to run on actual RTU hardware. Functional testing demonstrated that different users were able to carryout assigned tasks with the limited set of permissions provided by the security hardened RTU and a series of simulated insider attacks were prevented by the RTU role based access control system. Analysis of communication times indicated response times would be acceptable for many SCADA and DCS application areas. Investigation of a partitioning microkernel for an RTU identified the L4 microkernel as an excellent candidate. Experimental evaluation of L4 on real hardware found the IPC overhead for simulated critical RTU operations protected by L4 partitioning to be sufficiently small to warrant continued investigation of the approach.

TABLE OF CONTENTS

PAGE

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER I

INTRODUCTION

Supervisory control and data acquisition (SCADA) systems became popular in the 1960's for a variety of reasons. SCADA systems allow measurement and control of physical systems to be carried from a remote location. Initially they were used by industries and utilities to monitor and control physical devices like valves and switches. Prior to the use of SCADA systems, opening and closing of valves or the setting of switches was done manually; this was both costly because it was labor intensive and the exposure of valves and switches (especially in a distributed system like the electrical power grid or water supply system) to human control was considered a security and safety issue. Using SCADA systems, unauthorized access to valves and switches could be more tightly controlled while keeping a human in the loop; that is, human supervision and interaction were, and still are, part of SCADA systems. However, technological advances and the maturation of SCADA systems has pushed more of the supervisory function onto the computer systems that make up modern SCADA systems.

In the early development of SCADA systems attention was given to physical security, but virtually no attention was given to electronic or cyber security. The systems were obscure and the skills and technology needed to interact with the systems were simply not readily available; security of this type is often referred to as "security through obscurity". This pattern has continued and today "most dedicated SCADA and PCS applications have not included built-in security" [1]. Unfortunately, open protocols,

advanced telecommunication networks, cheap computer electronics, and unlimited access to even the most obscure information through the World Wide Web have made SCADA's security through obscurity obsolete. The move of SCADA systems to open standards and new technology has allowed SCADA system managers to realize cost savings by using commercial-off-the-shelf (COTS) hardware and software. In addition, as computer networks and information systems have become more commonplace throughout the corporate enterprise, managers have seen the economic benefits of having access to SCADA data and have built network connections into the previously isolated SCADA networks. The connection of porous and less secure corporate networks to once isolated SCADA networks, now using COTS systems, has unintentionally exposed SCADA systems to a host of vulnerabilities and threats for which it was ill prepared.

SCADA protocols provide no authentication or authorization capabilities. When other networks are connected to the SCADA network, intentionally or unintentionally, an attacker who manages to gain access to the SCADA network can spoof control signals on the SCADA network. Because SCADA protocols do not provide authentication or authorization a SCADA system is unable to distinguish between a real and a spoofed control signal, allowing the attacker to control SCADA devices. If the device were an electrical breaker and the SCADA operator was an electric utility, then turning that switch on might overload the power systems, or tuning it off might turn off electricity to customers. This threat is compounded by the use of COTS software, particularly COTS operating systems, as it becomes possible for insiders to use almost any PC to run SCADA software, and thus elevates the insider threat.

Concern for the cyber security of industrial control systems has been amplified by the fact that many, if not all, of our nation's critical infrastructures are heavily reliant on these control systems for reliable and stable day to day operation. The Patriot Act defines critical infrastructures to be "systems and assets, whether physical or virtual, so vital to the United States that the incapacity or destruction of such systems and assets would have a debilitating impact on security, national economic security, national public health or safety, of any combination of those matters" [2]. The President's Commission on Critical Infrastructure Protection found that there was a "growing cyber dimensions associated with infrastructure . . . and . . . the defenses that served us so well in the past offer little protection from the cyber threat" [3].

While the United States has been fortunate that a major cyber attack has not been successfully carried out against any critical infrastructure SCADA systems, incidents have occurred. In 2003 the slammer worm penetrated part of the network at a Davis-Besse nuclear power plant in Ohio and disabled part of the safety monitoring system for nearly five hours [4]. Fortunately the plant was shut down for repairs at the time. In another incident, a hacker using a radio transmitter was able to open valves and release raw sewage from an Australian sewage treatment plant [5]. The reluctance of companies to release incident information along with the possibility that some or many incidences go unnoticed makes it difficult to accurately assess the risk. One attempt to track incidents, the Industrial Security Incident Database maintained by the British Columbia Institute of Technology (BCIT), has shown a sharp increase in security incidents beginning in 2001 [6].

Addressing cyber security for SCADA is an ongoing task with many challenges. One challenge is that these systems tend to have a very long deployment life, up to and even beyond twenty years; consider the difference in computing technology between today and twenty years ago. Addressing the security needs for next year is challenging, addressing the security needs two decades into the future is daunting at best. Economics also plays a role because the cost of updating or replacing SCADA systems is significant, meaning that security solutions for legacy systems are needed. However, control systems are gaining in popularity; the global revenue from the sale of control systems is expected to grow to $13.9 billion by 2009 [1]. As this growth continues and as network convergence becomes an increasingly un-avoidable reality, it is of utmost importance that the next generation of SCADA systems be security hardened against all types of cyber-based attacks.

The SCADA architecture is generally broken down into a master station or MTU used by human operators to monitor and control remote terminal units, or RTUs. A communications network provides communication channels between MTUs and RTUs. Security hardening techniques are needed for the various components as well as for the SCADA system as a whole. RTUs interact with physical devices like valves and switches. A primary SCADA security objective is to prevent unauthorized or improper operation of valves, switches, or other physical devices, since these devices could have economic consequences for a SCADA operator as well as potentially disrupting normal operation of U.S. critical infrastructures. The fact that RTUs can, and often are, physically remote makes securing them that much more important.

This dissertation describes research and development of a security hardened RTU. While protecting and securing existing systems is important, the aim of this dissertation is to explore the development of next generation RTUs. As existing RTUs are replaced in existing SCADA deployments and as new SCADA systems are deployed, it is important that these RTUs be security hardened against cyber based attacks. This dissertation presents an RTU role based access control model for hardening RTUs. The model is developed to prevent unauthorized alteration of analog and digital IO points. In addition a middleware layer deployment architecture is advocated to allow fine grained and homogenous application of an RTU access control policy. Operating system (OS) support for a middleware layer deployment is a critical factor in the assurance of the security hardened RTU. Two approaches for reduced kernel RTUs are presented. A reduced commercial-off-the-shelf (COTS) kernel is one approach, and is used in the development of a prototype for testing. A second approach is to use a microkernel which supports partitioning and partition RTU software components to improve security.

1.1 Organization of Dissertation

Chapter two provides background information in SCADA and SCADA cyber-security as well as role based access control models and microkernels. Chapter three presents a high level description of the hardened RTU approaches which are investigated in this dissertation. Chapter four described the RTU role based access control model in detail. Chapter five discussed middleware layer deployment, and the role of an RTU operating system in security. Chapter five presents two reduced kernel RTU approaches, a reduced COTS kernel and a microkernel. Chapter six describes the development of a prototype hardened RTU based on the SIXNET mIPM and a reduced Linux kernel.

Performance and security testing results are presented in chapter six as well. Chapter seven describes additional investigation of microkernels and the results of some preliminary development and testing using an XScale PXA 255 processor and the OKL4 microkernel. Chapter eight presents the conclusion of this dissertation and elaborates on directions for future research.

Chapter II

LITERATURE SURVERY

This chapter presents background information of several topics relevant to this dissertation. Section 2.1 gives a historical overview of SCADA systems, their central components, and uses. The security threats and vulnerabilities that face SCADA systems are discussed in section 2.2. Section 2.3 presents a survey of SCADA security research and existing research challenges. Section 2.4 described the role based access control model (RBAC), and a presents a survey of the work on RBAC and constraints. Section 2.5 discusses the trusted computing base (TCB) of a system, security kernels, microkernels and separation kernels.

2.1 SCADA systems

Supervisory Control And Data Acquisition (SCADA) came into existence in the mid 1960's coinciding with the development of the minicomputer. SCADA provides a means for remotely monitoring and controlling many kinds of industrial systems by providing users of the system with the ability to remotely control one or more specific devices and to monitor the performance of those devices from a central and physically remote location. The IEEE std C37.1-1994 [7] defines SCADA to be:

"*A system operating with coded signals over communication channels so as to provide control of RTU equipment. The supervisory system may be combined with a data acquisition system by adding the use of coded signals over communication channels to*

acquire information about the status of the RTU equipment for display or for recording functions." [7].

An excellent example of such a SCADA system is the distribution system used by electric utilities, which is one of the oldest and most familiar SCADA systems. In electricity distribution SCADA is used to collect information from remote parts of a power distribution grid; for example the volts, amps or phase angle of a particular line in a substation, and provide it to a central control installation. In addition, SCADA allows an operator at the centralized control station to trip breakers at remote substations in response to conditions reported by the SCADA system. Other well known industries that use SCADA are the gas and oil utilities and nuclear power production.

2.1.1 SCADA Components.

There are four main components that make up a SCADA systems: the supervisory system or master terminal unit (MTU), remote terminal units (RTU), a communications network, and field instruments or devices [8-10]. The exact nature of the different components depends greatly on the specific SCADA system and its topology. A typical supervisory system is shown in figure 2.1 and each subsystem is explained in detail in the following paragraphs. A small SCADA system might consist of only one MTU and one RTU, and is referred to as single-master, single-remote [11]. A more common configuration is the single-master, multiple-remote system with a single MTU connected to many RTUs. In large SCADA systems it possible to have multiple MTUs and hundreds of RTUs [11].

Master station (MTU)

The master station or master terminal unit (MTU) has traditionally been located in a control room where human operators interact with the system through a user interface (UI). The MTU is responsible for polling remote devices for data, processing the data, providing various representations of the data (including alarms) and sending operator initiated control signals back to the field devices. In some situations the UI is carried out by a separate system called a HMI (human machine interface) system. The HMI system provides an interface between an operator and the MTU, freeing up the MTU from providing a UI. In this case the MTU continues to carryout polling and control activities, but the high level representation is left to the HMI machine. A sample operator screen typical of an HMI or MTU display is shown in figure 2.2.

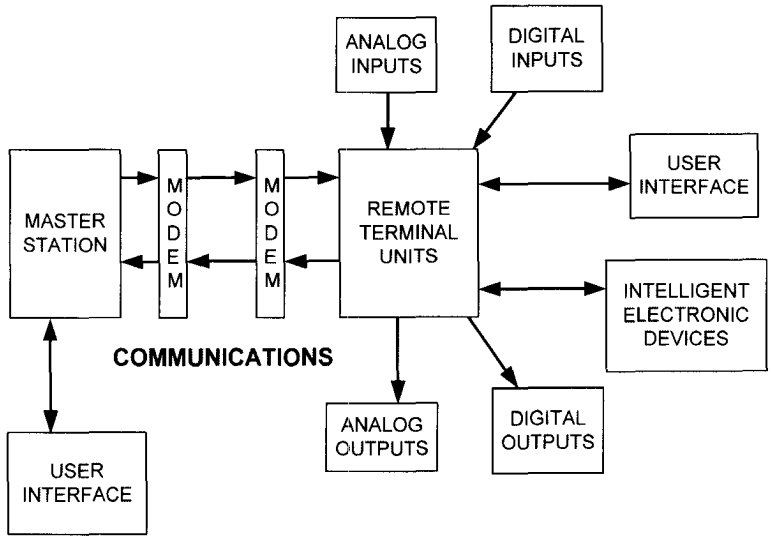


Figure 2.1. Typical supervisory systems [11].

Remote Terminal Units

Remote terminal units (RTUs), also referred to as remote telemetry units, are standalone systems that can acquire data from devices or equipment at the remote site, control devices or equipment at the remote site, and transfer acquired data back to a

master station. RTUs are typically built to withstand the much harsher operating environments that can be associated with remote locations like a plant floor, or an electric utility substation. RTUs provide four basic types of connections for interfacing with field devices: analog inputs, analog outputs, digital inputs, and digital outputs. Leads from field devices are directly connected to these interfaces on the RTU. An RTU also includes some communications capability through a combination of serial ports, built in modems, and more recently Ethernet ports. Other RTU components include a CPU, memory, power supply with battery backup, watchdog timer, surge protection, and real-time clock. A sample RTU specification is given in appendix A and figure 2.3 shows a generic RTU hardware configuration.

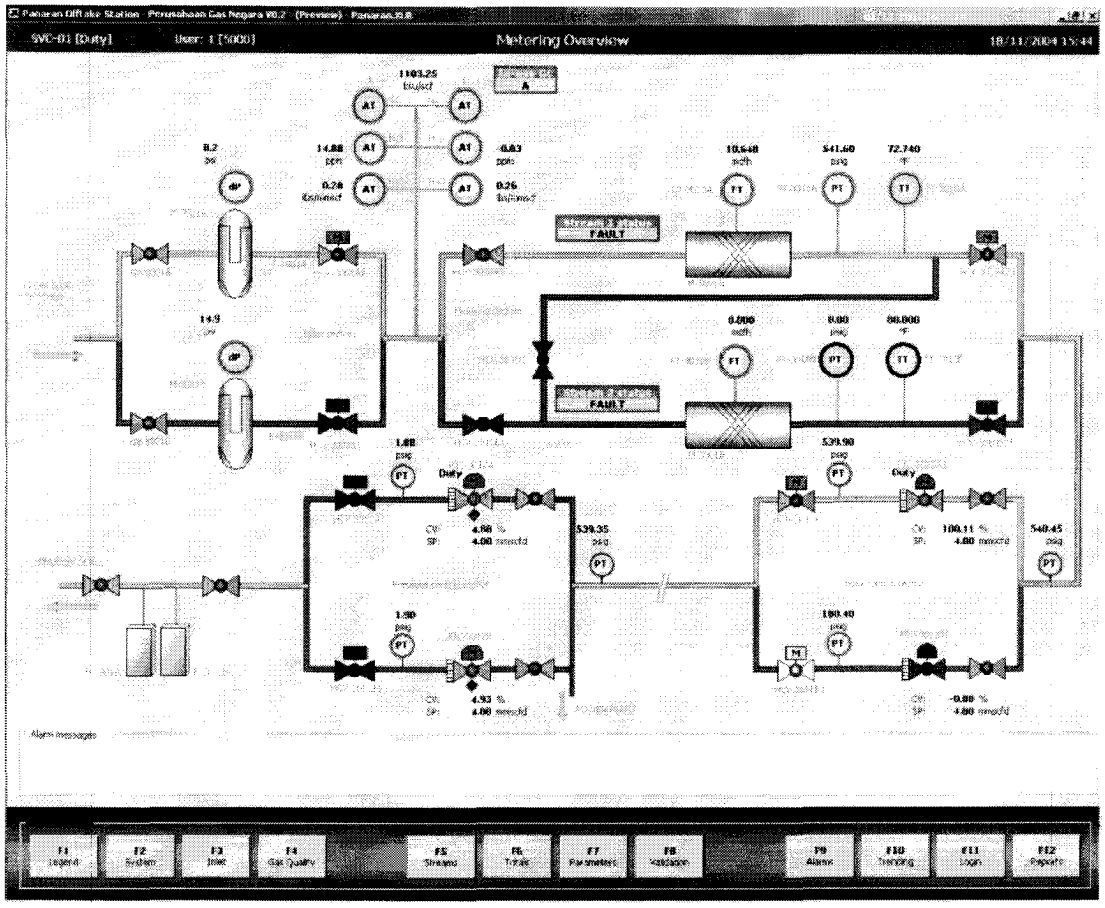


Figure 2.2. Sample SCADA HMI from [12].

Communications Network

The communication network of a SCADA system connects RTUs with MTUs. Remote locations may have a communications network, like a LAN, which can be used for local inter-device communication, but this is usually not considered to be part of the SCADA communications network. Communication links take many forms including leased lines, Public Switched Telephone Networks (PSTNs), Internet Protocol (IP) based landlines, radio, microwave and even satellite. SCADA communications security has traditionally referred to error detection and error correction capabilities, and not to features such as authentication and encryption [7;9].

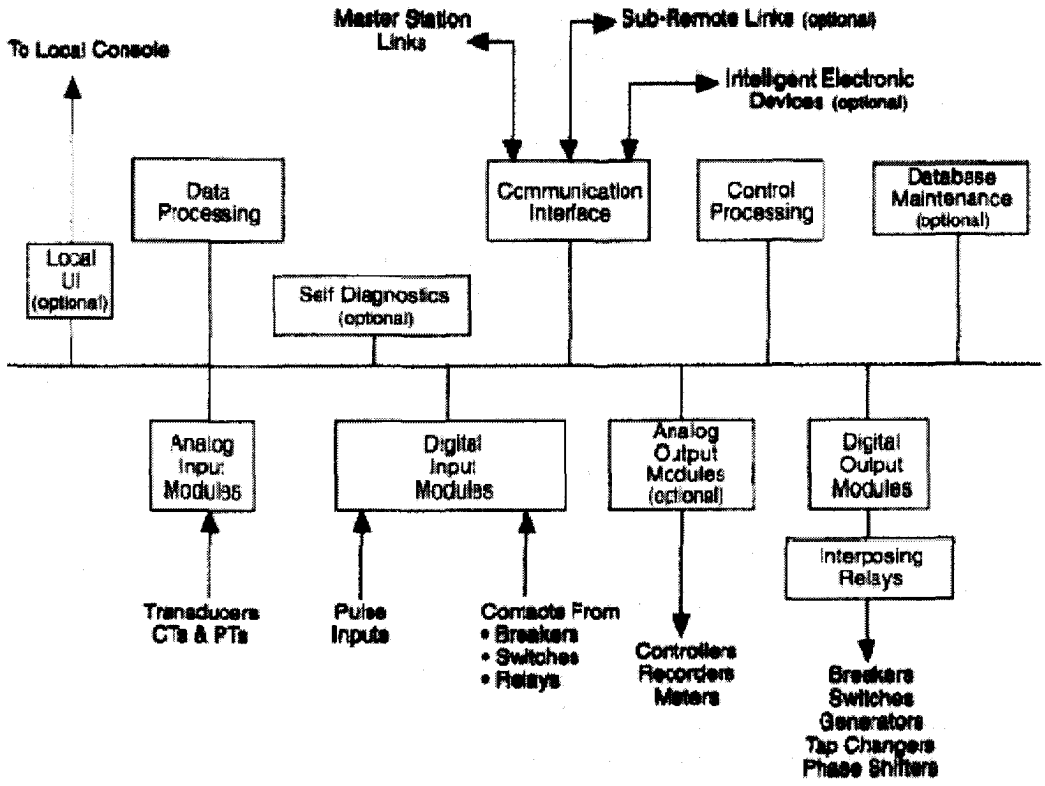


Figure 2.3. Basic RTU hardware components [13].

Field Equipment

At the periphery of SCADA systems are field equipment or field devices. These are the actual hardware components, which effectively serve as the eyes, ears, and hands of the SCADA system. Field equipment essentially consists of sensors and actuators. Sensors directly measure a physical condition at some remote site and actuators open, close, activate or inactivate a remote physical device. Some examples of field equipment are: voltage sensor, phase sensor, circuit breaker, relay, temperature sensor, pressure sensor, and flow control valve.

2.1.2 SCADA architectures

As computer and network technology have evolved and matured, so have SCADA systems. The evolution of SCADA systems is generally broken down into three separate successive generations [14;15]: monolithic, distributed, and networked. The changing architecture of SCADA systems has been a contributing factor to the cyber security issues faced by modern SCADA systems.

First Generation: Monolithic

At the time that SCADA systems were first developed, the mainframe computer was the dominant computer technology. Networks were virtually non-existent making mainframes standalone machines. The SCADA systems of this era reflect this paradigm. They were special purpose standalone systems that were not intended to be connected to other systems and tended to be very hierarchical and centralized in nature. Figure 2.4 shows a standard first generation SCADA architecture. The master station in these SCADA systems was typically a single mainframe computer. A second redundant master

station was usually present and shared the communications bus with the active master station. In the event of a system failure the second system could take over.

The lack of network technology led vendors of SCADA systems to develop solutions that allowed RTUs to communicate with the MTU mainframe often over long distances. The communication technology they developed was driven solely by this goal and in the absence of any of today's WAN protocols. In general the communication protocols developed by different vendors were lean, supporting only the minimal functionality needed to achieve scanning and control of points within a remote device [14]. The transmission medium used to connect RTUs and MTUs lacked a high degree of fidelity, leading to communication security focused exclusively on error detection and error correction codes. In addition each vendor tended to view their protocols as proprietary, preventing other vendors from developing equipment that could communicate using these protocols [15].

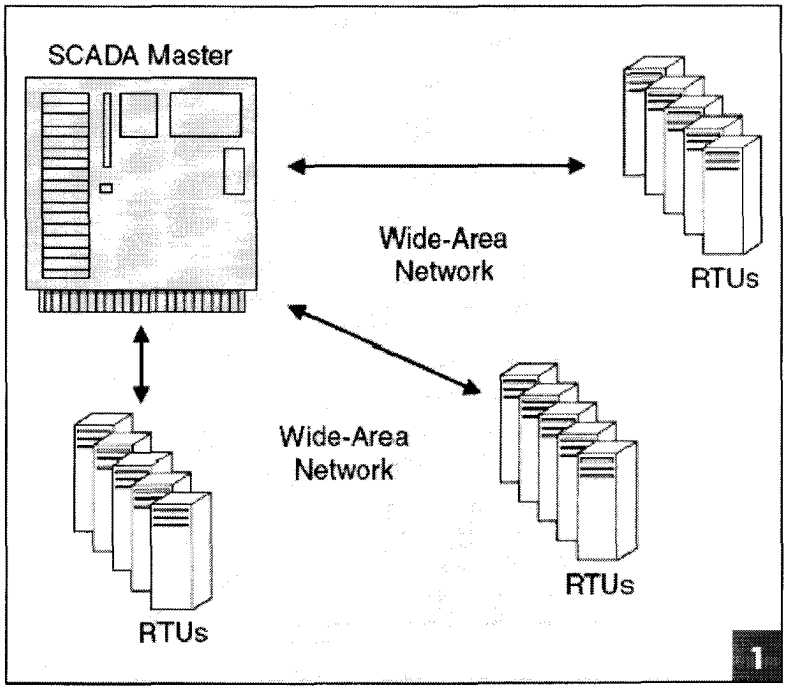


Figure 2.4. First Generation SCADA architecture [15].

Second Generation: Distributed

Advances in system miniaturization and LAN technology characterize second generation SCADA systems. The single mainframe master station was replaced by multiple stations serving different functions all connected by a LAN. The distribution of system functionality across multiple machines increased the overall processing capability of the system, but LAN technology was only capable of handling relatively short distances, typically hundreds of feet, this meant that the systems still had to be housed within a single room. Off-the-shelf LAN protocols were available, but some vendors still choose to use propriety protocols. Communication links with RTUs were largely unchanged relative to first generation systems, and in general vendors maintained control over what hardware, software, and devices were available for a specific SCADA system.

Third Generation: Networked

Third generation systems are similar in many ways to second generation systems, but with one important difference, which is the move to an open system architecture instead of a vendor controlled proprietary environments [15]. Open standards have removed the limitations that proprietary protocols placed on SCADA systems and therefore make it much easier to use COTS (commercial-off-the-shelf) components to build SCADA systems. One consequence of this move has been the use of WAN protocols like TCP/IP for communication between SCADA components like master stations, RTUs, field communication equipment, and HMIs [15]. Figure 2.5 shows a typical third-generation SCADA architecture. Some advantages of internet based SCADA systems are discussed in [16]; the primary advantage cited is lower costs.

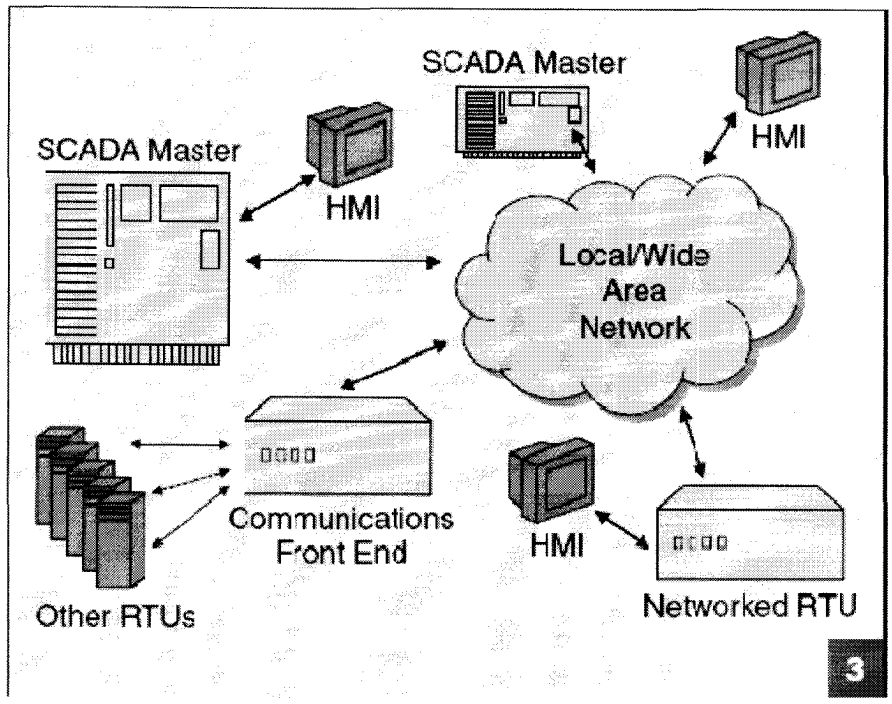


Figure 2.5. Third-generation SCADA architecture [15].

2.1.3 SCADA protocols

At the heart of SCADA networks are SCADA protocols. These provide the template for communication between SCADA components, typically between the MTU and the RTU. Early SCADA systems, the first and second generation SCADA architectures discussed previously, used proprietary protocols, but in more recent years there has been a move to open standards in SCADA protocols. RTUs are connected to MTUs by a variety of different communication channels and both the cost and availability of the communication channels has affected protocol design [17].

The limited bandwidth of early communication channels resulted in a very compact message format, supporting only the most basic information needed to achieve RTU to MTU communication. Figure 2.6 shows the structure of the basic SCADA message format. The four bit RTU address allows multiple RTUs to share a single communication channel, rather than requiring a separate communication channel for each

RTU. The eight bit function code specifies what operation is to be performed by the RTU. The bits following the function code are an addressing scheme that indicates the set point, control point, or data on which the operation is to be carried out. This address has no special meaning to the RTU, and it is up to the MTU and SCADA software to correctly associate an RTU address with the real world value it represents. According to the American Gas Association's AGA-12 standard there are about 150-200 SCADA protocols [18]. Some of the more popular SCADA protocols, as shown in table 2.1, are: MODBUS, IEC 60870-5-101, and DNP3, but none of these currently contain security features [19].

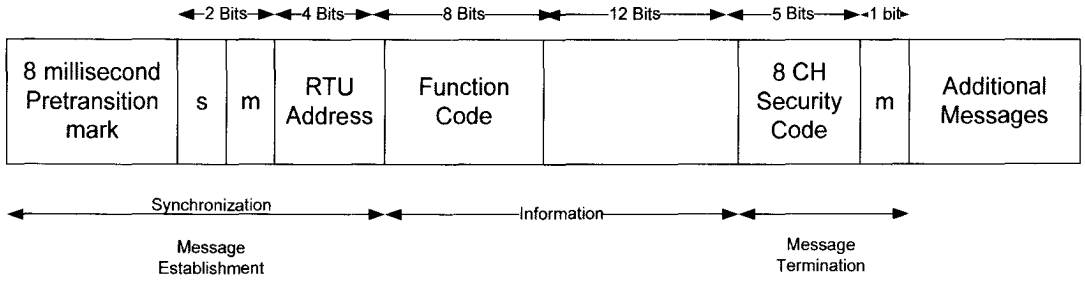


Figure 2.6. Typical SCADA protocol message format, adapted from [17].

2.2 SCADA cyber security threats and vulnerabilities

The primary cyber based threat to SCADA systems is that an unauthorized person or agent will access the SCADA system and interfere with its operation. The IEEE guide for Electric Power Substation Physical and Electronic Security defines an electronic intrusion as:

"*Entry into the substation [RTU] via telephone lines or other electronic-based media for the manipulation or disturbance of electronic devices. These devices include digital relays, fault recorders, equipment diagnostic packages, automation equipment, computers, PLC, and communication interfaces.*" [20].

Typical attack scenarios like those described in [21;22] center around an attacker making changes to control settings, physical device parameters, or sending control commands directly to field devices. These attacks would result in a malfunctioning of the SCADA system which might cause a disruption in service, or possibly environmental damage or loss of human life. These threats might be carried out by a number of potential threat agents, including hostile nation states, industrial spies, disgruntled employees, and malicious hackers. Table 2.2 lists possible threats to SCADA systems.

Table 2.1. Common SCADA Protocols [23].

Protocol	Organization	Common Industries	Features
DNP3	Developed by GE Harris, Managed by the DNP organization	Electric Utilities, Gas distribution, and Water distribution	Object Oriented. Three layer OSI model. Open non-proprietary standard.
Modbus (Modbus/TCP)	Developed by Modicon	Gas and Oil and electric substations, transportation	Initially developed for modicon's PLCs. Is an open standard and is royalty free. Simple to implement. Both serial and TCP version are available. Simplicity and wide use make this an excellent protocol when integrating multiple application.
Ethernet/IP (Industrial Protocol)	Open DeviceNet Vendors Association (ODVA)	Industrial Automation	
Device Net	Open DeviceNet Vendors Association (ODVA)	Industrial Automation	Uses CAN as its backbone, originally developed by allen-bradley. Supports master-slave as well as peer to peer
IEC 60870-5	IEC TC57		
IEC 61850	IEC TC57	Substation automation, distribution automation	Ultra fast response times

One of the most serious vulnerabilities faced by SCADA system is the commonly held misconception that control networks are isolated and therefore not accessible to attackers [24;25]. Early control systems used a combination of knobs, lights and dials mounted on specialized custom-built control panels. Communication with process machinery and field equipment was achieved using analog control signals carried by dedicated cables that connected the process control panels to field equipment [26]. Securing these systems was simply a matter of locking the door to the control room. The first major technological change affecting the cyber security of control systems was the adoption of digital communication through serial networks and the ubiquitous RS-232, RS-422 and RS-485 standards. At this point, networks, often proprietary or leased serial lines, were still relatively isolated. However, the use of digital communication created a consolidation of both communications channels and communication standards [26]. As computers and network technology began to become available and used through out the enterprise, there has been increased demand by industry for connection between the plant floor and the corporate network. At the same time there has been increased public availability of network access and computer technology. As a result, there is now almost always the possibility of an external connection being able to reach the control network, whether through an intranet, a business partner's networks, or the Internet. In addition to these standard network paths, many SCADA systems make use of modems to provide connectivity which can also allow an external connection into the SCADA network. For example, the use of war-dialers to connect to remote SCADA equipment is described in [21;22]. The assumption that SCADA networks are isolated and therefore protected from potential attack is simply not true today [25;27;28].

Table 2.2. Threats to SCADA systems.

Threat	Description
Hackers	Hackers break into networks and systems for the thrill and challenge that it presents. SCADA systems are not exempt, and are now receiving the attention of hackers (http://www.msnbc.msn.com/id/20128089/).
Hostile Nation States	Because they control critical infrastructures on which we are dependent SCADA systems are an excellent target for cyber warfare.
Foreign Intelligence	Intelligence agencies (foreign and domestic) are using cyber tools as part of there intelligence gathering capabilities. Attacking SCADA systems could provide intelligence was well as feed information into offensive branches.
Botnets	Botnets are a collection of compromised computers controlled by single person, usually referred to as a bot-herder. Botnets are used to carryout coordinated attacks, send spam, or carryout phishing schemes. Botnets make use of automated attack software. Botnets present two threat vectors, one they can be used to carryout an attack on SCADA systems, or two, SCADA systems may become part of a botnet and have their resources depleted by the botnet activities.
Insiders	Disgruntled insiders have been main source of computer crime since they have knowledge of and access to internal systems. Insiders include employees, business partners and vendors. Insiders may not necessarily be malicious, but accidental mistakes can have the same consequences as malicious attacks.
Worms	Worms are automated programs that propagate themselves though networks by exploiting a common vulnerably. Worms can exhaust network and computer resources, as well as harm files on the victims.
Viruses	A Virus is a program that can replicate itself and pass on malicious code to other non-malicious programs. Viruses can corrupt files and disrupt or interfere with the normal operation of a computer system.
Terrorists	Terrorist seek to destroy or incapacitate critical infrastructure in order to damage public moral. Cyber attacks on SCADA systems are one way to achieve this and may be possible from a point of relative obscurity. Cyber attacks on SCADA systems may also be used to leverage a physical attack, for example by hiding alerts of a malicious physical attack.
Industrial Spies	Seek to acquire trade secrets, or inside knowledge that can give one organization advantage over another. SCADA systems in manufacturing industries will have knowledge of trade secrets, or just private status data. Corruption of a competitor's SCADA system at the appropriate time could have financial benefits for the competitor.

As mentioned in the previous section, early SCADA installations were characterized by closed systems and proprietary protocol standards. Most SCADA systems are privately owned and operated, and operators are driven by economic forces. For these reasons the economic advantages offered by open standards and open architectures has strongly motivate the adoption and integration in SCADA. In addition to assumption the SCADA networks were isolated, was a widely held belief that it was difficult to acquire information about SCADA system [6;27]. Open standards and open

application layer interfaces that make use of available commodity software, such as a web interface. These additional application layer interfaces in to device introduce additional vulnerabilities and attack vectors into SCADA systems.

A final SCADA vulnerability comes from the increased data exchanges between businesses achieved through network connectivity. For example, deregulation in the power industry has created vulnerabilities for electric power generation, transmission, and distribution SCADA systems. As a result of deregulation, data exchanges between single vertically integrated organizations have been replaced by many horizontal relationships among independent entities [15]. Some of the vulnerabilities that result from deregulation are described in [32]. The complex interaction among entities not only increases the network connectivity of SCADA systems but can require multiple master and multiple remote architectures with many different entities needed varying degrees of access.

Evidence of the vulnerabilities faced by SCADA systems is well documented in a recent assessment of the network security of power substations [33]. In this assessment Oman and colleagues found a number of security vulnerabilities, identified in 1997, still existed in 2002. These included such basic security vulnerabilities as default passwords and unsecured modem access. They also found new potential vulnerabilities in the form of internet connectivity and wireless networks. Table 2.3 summarizes the vulnerabilities they identified.

Table 2.3. Power substation vulnerabilities [33]

Documented Vulnerability	1997 NSTAC	2002 Visits
Weak Passwords Used	✓	✓
Default Passwords Not Changed	✓	✓
Passwords Posted Visibly	✓	✓
Shared Logins	✓	✓
Inconsistent or Non-existent Warning Banners	✓	✓
Personnel Unaware of Hacking Threat	✓	✓
Non-existent Security Policies	✓	✓
Unsecured Modem Access	✓	✓
IT Network Interconnectivity	✓	✓
Non-existent or Inadequate Intrusion Detection	✓	✓
Internet Connectivity	Non-existent	✓
Wireless Networks	Non-existent	✓
Commercialization of Utility Telecomms	Non-existent	✓

There is also evidence that actual attacks against SCADA systems are occurring and that the number of attacks is increasing. A study by the British Columbia Institute of Technology [6] found a substantial increase in the percentage of attacks coming from external sources. BCIT maintains an industrial cyber security incident database for the purpose of tracking cyber security incidents in process control systems. They found a substantial increase in the number of attacks beginning in 2001. An analysis of incident type found that between 1982 and 2000 about 31% of the incidents came from external sources but that from 2001 to 2003 nearly 70% of the incidents came from external sources. Further analysis of the external security incidents to identify entry points concluded that there are many routes into complex SCADA systems.

2.3 Securing SCADA systems

Having established and understood the weak security of modern SCADA systems the question then becomes how to secure them. An obvious first step is to attempt to apply established network security technologies to SCADA networks. Section 2.3.1 discusses a number of articles that explore applying standard IT security solutions to SCADA systems. However SCADA systems and traditional IT systems are not the same, and care must be taken when applying existing security technologies to SCADA since these technologies, which acceptable in traditional IT environment, may have unacceptable adverse impacts on SCADA. Section 2.3.2 discusses the difference between traditional IT environment and SCADA or control networks. In cases where traditional IT solutions are not feasible, new security technologies need to be developed to address the specific needs of SCADA. Section 2.3.3 presents the research challenges facing SCADA and the work that has been conducted.

2.3.1 Securing SCADA with standard IT technologies.

Applying the experience, knowledge, and technologies of IT security to SCADA and PCS systems has been an essential first step in securing SCADA systems. As we have seen, the security threat to SCADA systems comes in a large part from the fact that these were once isolated networks. When they can no longer be isolated, good network segmentation can help keep SCADA systems secure [22;34;35]. Segmentation can be provided by firewalls or through the use of a virtual LAN (VLAN) [36;37]. Network segmentation reduces the exposure of SCADA systems to external networks, improving security.

In [38] Munshi discusses the security considerations for SCADA systems at four levels of the SCADA architecture. Level one is field equipment like PLCs and RTUs. The threat identified at this level is access to data or spoofing of commands, and the recommended solution is to implement encryption. Level two is the telecom level comprised of the communication channels used to connect RTUs and field equipment to level three. The threat at this level is that these are generally unsecured communications that may be traveling over unsecured shared networks. The recommendation at this level is to consider using IPSEC. Level three is the SCADA level, essentially this is the control center. Recommendation for systems in this level include operating system hardening, patch management, network equipment access control, server access controls, physical security, virus protection strategy, and user authorization. The final level, level four, is the enterprise level, consisting of remote SCADA clients, ERP systems, corporate users of SCADA data, web services, and so forth. Recommendations at this level include network controls like firewalls, proxy servers, and network segmentation.

A layered security approach is advocated by Miller [30]. Layered security deploys security elements in each of three layers of a computing environment, personnel, network, and operating system. Each layer includes some form of examination, detection, and prevention. According to this model, the SCADA computing systems are segmented and compartmentalized based on functional groups and access control plans. Access control matrices are developed that provide a detailed security policy, which is then implemented using security products for examination, detection, prevention, and encryption at the various layers.

As previously mentioned, the use of modems for remote access to SCADA systems provides an easy target. Abshier and Weiss [34] suggest keeping modems unplugged when they are not needed. In situations where this is not possible the use of dial back modems is recommended. Password protected modems and encrypting modems are another possible solution suggested by Oman, Schweitzer, and Roberts [21].

Vulnerability assessment tools like Nmap, Nessus, and Ethereal have become standard in the IT security community, Permann and Rohde [39] discuss the use of these tools for security assessment of control systems. The fact that the behavior of SCADA devices may be unpredictable when scanned makes them of limited value. For example, non-aggressive network scanning by Brown [26] caused the failure of PLCs from two different manufacturers. Recently, Tenable and Digital Bond have worked together to develop SCADA Nessus plug-ins for control system vulnerability scanning [40].

Oman, Schweitzer and Roberts [21] give an extensive list of mitigation technologies and tools from password generators and biometric devices to firewalls, intrusion detection systems and public key infrastructure. They also provide an extensive list of recommendations and best practices. Some recommendations not already mentioned include the use of two or even three factor authentication when appropriate, avoid using the same password for multiple systems, use warning banners to discourage electronic intrusions, limit the number of failed login attempts allowed for a single connection. Similar recommendations are also given in [32;41].

An extensive case study of a secure substation information system installation using standard IT technologies is described by Dolezilek, Carson, Leech, and Streett [42]. The system provides comprehensive multilayered security integration and

combines both the SCADA network and the business network. Another case study, of a PCS for a pulp mill in Canada, is described by Byres in [43] and focuses mainly on network segmentation.

In addition to the use of specific security mechanisms and technologies for securing SCADA systems, improved management strategies and processes are also needed. Abshier [44] summarizes ten important design and process principles for securing control systems. The principles are: governance, security awareness and training, policies and procedures, change management, security architecture, adding devices and remote access, vulnerability, risk assessment and penetration tools, incident response, configuration and patch management, and monitoring. The goal of following these principles is to ensure that due diligence has been followed in securing an organizations control systems. Some additional strategies for building a security plan are given in [45].

Comprehensive guidelines for creating secure SCADA and control networks are also being developed by several industry organizations. These documents provide guidelines for establishing secure SCADA systems through definitions and best practices; in some cases specific technologies are discussed, but in others only the desired result is given. Many of these documents are still under development and review, but public drafts are available.

Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security (SP 800-82) [46]

SP 800-82 is one of the products of the National Institute of Standards and Technology's (NIST) initiative on critical infrastructure protection called the Process

Control Security Requirements Forum (PCSRF) [47]. The forum is a working group of over 500 members that come from government, academia, and the private sector. The main goal of the PCSRF is to increase the security of industrial process control systems through the definition of a common set of standards. The PCSRF has also published the first draft of its System Protection Profile for Industrial Control Systems (SPP-ICS) [47] and field device protection profile [48].

ISA-SP99

ISA, a nonprofit organization concerned with standards in industrial automation, is in the process of developing ISA-SP99, a standard for manufacturing and control system security. The goal of ISA-SP99 is to establish standards, practices, technical reports and related information for implementing electronically secure manufacturing and control systems.

NERC CIP 002 – CIP 09

The North American Electric Reliability Council or NERC, is self-regulatory organization that sets standards for reliable operation of the bulk electric system. NERC's members come from all segments of the power industry: investor-owned utilities, federal power agencies, rural electric cooperatives, state municipal utilities, and independent power producers. NERC is in the process of drafting cyber security standards to reduce the risk of a cyber compromise of bulk electric systems. The first draft was known as NERC 1300 and was issued in September 2004. In response to comments on the first draft a second draft was written. The seconds draft, renamed NERC CIP is organized in eight sections, CIP – 002 through CIP – 009. CIP – 005

focuses specifically on electronic security, defining electronic perimeters, and requirements for logging, password management, access control, and strong authentication.

2.3.2 Differences between SCADA and Traditional IT environment

Though SCADA systems are increasingly adopting technologies from traditional IT environments, SCADA and traditional IT systems are very different in several ways. One of the most important differences is how security is prioritized. In traditional IT systems, security engineers usually consider confidentiality the most important followed by integrity and then availability. However, as discussed by Miller in [30], for control systems availability is most important, followed by integrity and then confidentiality. For example, when you switch on a light, it needs to come on; when you pick up the phone there should be a dial tone. This is availability; it is what we expect from the systems and services that make use of SCADA. Moreover, down time for the services that SCADA systems operate can run into the millions of dollars per hour [30], making availability of paramount importance.

SCADA systems and other control system also tend to have very different performance needs from traditional IT systems. Delaying the delivery of information even for a relatively brief moment is not acceptable in SCADA systems, though they often do not require a high degree of throughput. However, IT systems typically do require a high throughput but are much more tolerant of delays or jitter. In addition many SCADA systems may have much greater resource constraints than would be found in traditional IT systems. This lack of computing resources along with performance constraints can make it difficult or impossible to apply standard security technologies.

One of the real challenges presented by SCADA system is the relatively long life of SCADA components compared to their IT counter parts. It is not uncommon for SCADA components to be in use for fifteen to twenty years, while the average IT system as life span of three to five years.

Table 2.4. Difference between SCADA and Traditional IT

Category	Information Technology Systems	SCADA Systems
Performance	High throughput, can tolerate delay and jitter	Medium to low throughput but cannot tolerate delay or jitter
Focus of security architecture	Protect focus on central core of the system. So called Hard in the middle and soft on the outside	Need to protect the edges or perimeter devices such as RTUs and field devices. Also need to protect core internal systems as well
Priority of security primitives	1. Confidentiality 2. Integrity 3. Availability	1. Availability 2. Integrity 3. Confidentiality
Component lifetime	3 – 5 years	15 – 20 years
Physical accessibility	Easily accessible	Isolated and remote, may be very difficult to access

2.3.3 SCADA research challenges and current research.

Over the past several years industry groups and academics have begun to work towards addressing the SCADA security issue. This can be seen in the increasing number of publications related to SCADA security [1;18;30;49]. Igure [18;27] identifies three research challenges in the field of SCADA security. The first challenge is to improve access controls to SCADA networks to make it harder for attackers to gain access to the SCADA network. The second challenge is to improve security inside SCADA networks, including developing efficient monitoring tools that make actually carrying out an attack difficult. Finally he points out the need to improve the security

management of the SCADA network. Solutions to these challenges must take into consideration the unique demands of SCADA systems discussed in section 2.3.2.

One of the primary security tools is encryption, and there are several articles which present SCADA security solutions that deal with encryption. Leading the way in SCADA and encryption, particularly for legacy systems is the AGA 12 working group established in 2001 by the American Gas Association (AGA). The working group was to recommend solutions to that would help protect gas utility SCADA equipment from cyber attack. The group determined that unprotected serial based communication channels posed the greatest threat. In response to this threat AGA 12 has developed a serial SCADA protection protocol (SSPP) which is implemented by a separate device called SCADA Cryptographic Modules (SCM); these are installed on either end of a communication channel [50-52]. Figure 2.7 shows the proposed architecture of using two SCMs to provide encrypted communications between an MTU (SCADA Host) and an RTU.

Wright, Kinast and McCarty [52] present a low-latency encryption scheme for retrofitting serial SCADA communications. This proposed solution attempts to fulfill the requirements of a SCM specified by AGA -12. Recall that an SCM sits on either side of a SCADA communication link, invisibly encrypting and decrypting all communications between an MTU and an RTU. Wright, Kinast, and McCarty describe two unique requirements for encrypting SCADA communication links. The first is that SCADA communication messages usually follow very predictable patterns, making plaintext attacks possible and likely. The second is that the real-time nature of SCADA systems means they can endure very little communication latency that will result from the

encryption and decryption of messages. The protocol prevents injection of unauthentic ciphertext, modification of ciphertext during transmission, reordering of messages, and replaying of old messages, while introducing a fixed latency of 2 * b/8 where b is the number of bits in a block. A different approach to meeting the AGA 12 SCM proposal is described in [19].

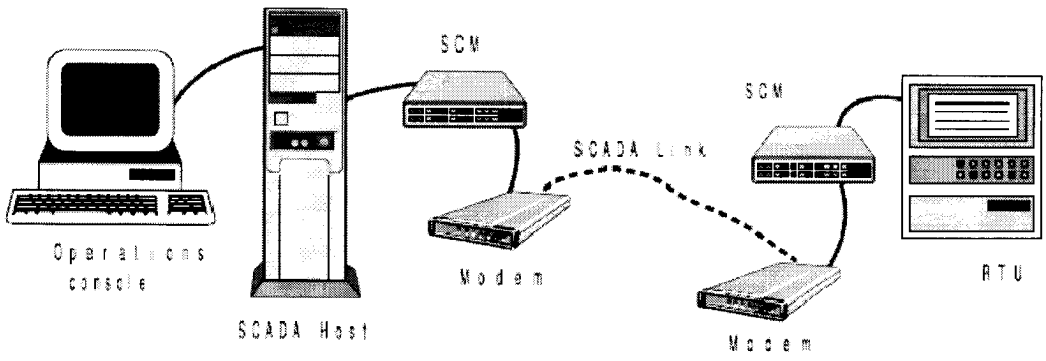


Figure 2.7. Encryption of SCADA serial communications with an SCM [52].

A secure SCADA protocol that addresses message integrity and sender authentication is presented by Patel in [29]. Several approaches are highlighted, such as SSL/TLS wrapping, the use of digital certificates, and the use of challenge response with a pre-shared secret. The DNP3 protocol is extended to include the necessary authentication objects so that RTUs or MTUs can use the proposed protocol to verify sender authenticity and detect modifications to messages. A threat analysis and formal proof techniques support security claims about the communication protocol. The focus of the protocol is on integrity of message and sender authenticity and is not concerned with confidentiality.

Some key management issues for a SCADA networks are investigated by Beaver and colleagues in [53]. They point out that many critical SCADA communications have

minimum time delays on the order of two to four milliseconds and therefore the processing time of public key protocols especially when run on the less powerful processors often found in SCADA devices prohibits the use of public key protocols for performing authentication unless highly specialized cryptographic accelerators are used. As an alternative to expensive hardware encryption they propose to let IEDs or field devices use symmetric encryption via a key exchange algorithm. They provide security for four communications paths: Master to Certificate Authority, Master to Substation, Substation to IED, and Substation to Substation. Communications are restricted to only these communication paths. They then describe a new format for packets within the SCADA network, and key generation, key storage, and key use associated with each communication path. Only substation to substation communications use public key algorithms, and then only for key exchange. They also describe a process for recovering from a substation penetration.

The use of smartcards to provide encryption and authentication to field area network nodes is described by Palensky and Sauter in [54]. A field area network (FAN) is a collection of nodes at some site, similar to a remote location. A FAN gateway provides remote access to the FAN much like an RTU provides remote access to field devices. The proposed security architecture is to place a smartcard with one or more pairs of public-private keys with each node. Data encryption is carried out by the smart card, and a corresponding smart card on the receiving end. In addition two access control schemes are described.

A DoS mitigation strategy for SYN flood attacks on SCADA systems is described by Bowen, Buennemeyer and Thomas in [55]. The approach they propose is based on

client puzzles. Client puzzles defend against DoS attacks by forcing clients, including attackers, to expend computational resources to calculate the solution to a puzzle, usually a cryptographic puzzle or hash function. Only after a valid solution is returned is a connection for the client created. Bowen et al. present a modified TCP protocol, called pTCP, that implements the client puzzle strategy. MTUs are the clients in this system and RTUs act as servers. When RTUs respond with a SYN+ACK to the MTU connection requests, a nonce and difficulty level is included in the response. The MTU uses the nonce to calculate a puzzle solution, with the difficultly level indicating the computational complexity of puzzle. As a DoS attack build (i.e. the number of established connections increases) the server increases the difficulty level, making the connecting stations and attackers commit greater resources to solve the puzzle. The goal is for attackers to cease committing resources to the attack before the puzzle difficulty level adversely impacts the delay of SCADA messages. Simulation using ns2 was done to evaluate the potential impact on the latency of SCADA messages. The focus of their simulation was on routine SCADA transactions, which they claim must have a delay time less than 540 milliseconds. The simulation found that for normalized difficulty levels below -9.5 latency increase was acceptable.

The use of standard intrusion detection systems was recommended by several articles in section 2.4.1, but Naedele and Biderbost proposed in [56] a human assisted intrusion detection system designed especially for process control systems. The proposed idea is to provide system security information in a form that does not require information security knowledge so that a process control system operator can monitor network security in the same way she might monitor a process. A prototype process control

system HMI for intrusion detection is presented where IDS alerts and alarms are presented in a combination of graphical and textual data to the operator in terms of the consequences or dangers they represent to the system. Another IDS framework proposed by Naess, Frincke, McKinnon, and Bakken [57] is designed for embedded systems in general, but the authors point out its potential for SCADA systems. Their proposal is a configurable middleware-based intrusion detection framework for MicroQoSCORBA.

Some methods for assessing the vulnerability of SCADA system are proposed in [58;59]. Byres, Franz, and Miller [58] describe the use of attack trees in assessing the MODBUS protocol. While the attack trees that were developed significantly improved the ability of the researchers in finding exploits, they were also useful in selecting an appropriate mitigation strategy. Conte de Leon et al. [59] describes a graph based model for calculating device vulnerabilities of SCADA systems. Each node in the graph is a device and device x is visible to y if there is a path from x to y and x and y are able to communicate through the physical network described by the path. A vulnerability value is assigned to each potential visibility path and a device vulnerability level can then be calculated by summing these values. The most vulnerable device has the highest vulnerability level. Additional research is needed in determining the initial assignment of vulnerability levels.

A generic SCADA security policy framework to assist in the creation of SCADA security policies is described by Young, Stamp, Dillinger and Rumsy [60]. The framework is organized in three hierarchical layers and supports detailed specific sub-policies that support generic high level policies. The framework was developed out of the author's experience in SCADA assessments and secure communication system

development and implementation. The policy they develop is broken down into eight main categories at level one, ranging from data security and personnel security to network security and physical access. The goal of the policy framework is to assist asset owners in the creation of SCADA security policies. Implementation details for two example cases are given to demonstrate the model.

Modbus/TCP, like most SCADA protocols has no security features and is unable to authenticate or authorize individual requests. A device using Modbus/TCP typically lacks packet filtering capabilities and therefore will carry out any legitimate command that reaches it. A common network security solution would be to filter the Modbus/TCP port as it passes through a firewall or router, enforcing an access control policy for device connection. However, this only allows access control at a source level, while some organizations' security policy may dictate that some hosts have read access to data, while other hosts have both read and write access to the device. An application layer filtering firewall is presented by Franz and Pothamsetty [61] that allows filtering of packets based on Modbus header values. This makes it possible to grant some hosts the ability to read from the Modbus slave device while not writing to it, and to other hosts the ability to both read and write. This work has been released under open source licensing and is available at `http://modbusfw.sourceforge.net/`.

An alternative architecture for the information and communication network of power systems is proposed by Xie, Manimaran, Vittal, Phadke, and Centeno [62]. The proposed architecture includes all the traditional elements of SCADA systems. The primary object of the architecture is to provide greater reliability through redundancy, though communication security is considered as well. Redundant communication

channels are combined with VPN and firewall technology to provide reliable but secure communications among entities.

Another next generation SCADA communications architecture is proposed by Hauser, Bakken, and Bose [63]. The proposed architecture, referred to as GridStat, is a middleware framework with API stubs that correlate with traditional SCADA functions polling, events status, and control settings. GridStat was designed to support flexible communications, making new types of controls and better situational awareness possible. GridStat also provides schemes for trust management, with the ability to approve new subscription, make routing decisions, and manage access control.

2.4 The Role based access control model

Role Based Access Control (RBAC) is an alternative access control model to the classical forms of access control that grew out of the access control matrix model [64] and the Bell LaPadula Model [65]. RBAC was proposed as a means of simplifying access control and including functional capabilities [66]. RBAC is policy neutral but it supports three well-known security principles: separation of duty, least privilege, and data abstraction. Interest in RBAC has continued to increase and lead to a variety of different models and extensions. In response to the lack of any widely accepted standard Ferraiolo has proposed a NIST standard for role based access control [67]. This standard serves as reference model for common dialogue on role based access control. The basic RBAC model for this dissertation will be the NIST model.

The standard defines three RBAC models, core RBAC and two extensions to core RBAC, hierarchical RBAC and constrained RBAC. Figure 2.8 shows the core RBAC model elements and their relationships as defined in [67].

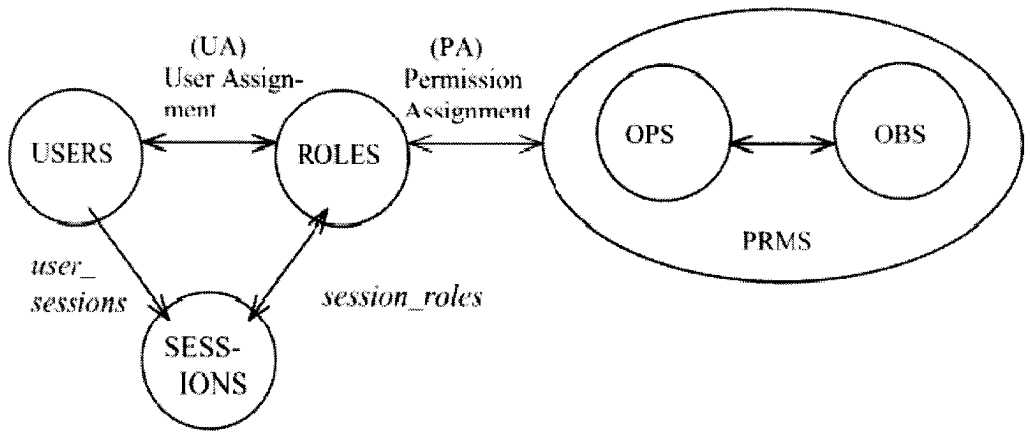


Figure 2.8. Core RBAC model [67].

RBAC consists of five sets: users (USERS), roles (ROLES), objects (OBJ), operations (OPS), and permissions (PRMS). The basic concept of RBAC is that users are assigned to roles and permissions are assigned to roles, allowing roles to serve as a mapping between permissions and users. This simplifies the assignment of permissions to users, more accurately reflects how organizations think about permissions, and greatly simplifies role revocation. A *user* is most often a human being, but the notion of user can be extended to other entities like devices, networks or autonomous agents. *Roles* attempt to approximate different job functions within the organizational construct in which the system is participating. A *permission* is the right to carryout an operation on one or more objects. An *operation* is some type of function to be carried out by the system for a user. *Objects* are entities that contain or receive information; their exact type depends on the system. Some examples of objects are files, directories, and database tables, rows, and columns. The purpose of applying RBAC to a system is to protect the system resources represented by objects.

—*USERS*, *ROLES*, *OPS*, *and OBS* (users, roles, operations, and objects, respectively).

—$UA \subseteq USERS \times ROLES$, a many-to-many mapping user-to-role assignment relation.

—*assigned_users*: $(r{:}ROLES) \rightarrow 2^{USERS}$, the mapping of role r onto a set of users. Formally: $assigned\_users(r) = \{u \in USERS \mid (u, r) \in UA\}$.

—$PRMS = 2^{(OPS \times OBS)}$, the set of permissions.

—$PA \subseteq PRMS \times ROLES$, a many-to-many mapping permission-to-role assignment relation.

—*assigned_permissions*$(r{:}\ ROLES) \rightarrow 2^{PRMS}$, the mapping of role r onto a set of permissions. Formally: $assigned\_permissions(r) = \{p \in PRMS \mid (p, r) \in PA\}$.

—*Ob*$(p{:}\ PRMS) \rightarrow$ {op $\subseteq$ OPS}, the permission-to-operation mapping, which gives the set of operations associated with permission p.

—*Ob*$(p{:}\ PRMS) \rightarrow$ {ob $\subseteq$ OBS}, the permission-to-object mapping, which gives the set of objects associated with permission p.

—*SESSIONS*, the set of sessions.

—*user_sessions* $(u{:}\ USERS) \rightarrow 2^{SESSIONS}$, the mapping of user u onto a set of sessions.

—*session_roles* $(s{:}\ SESSIONS) \rightarrow 2^{ROLES}$, the mapping of session s onto a set of roles. Formally: $session\_roles\ (s_i) \subseteq \{r \in ROLES \mid (session\_users\ (s_i), r) \in UA\}$.

—*avail_session_perms*$(s{:}SESSIONS) \rightarrow 2^{PRMS}$, the permissions available to a user in a session, $\bigcup_{r \in session\_roles(s)} assigned\_permissions(r)$.

Figure 2.9. Core RBAC definitions [67].

The flexibility and high degree of granularity of RBAC comes from role relations. The user assignment relation (UA) assigns different *roles* to each *users*. The permission assignment relation (PA) assigns various permissions to each role. The result of these two assignment relations simplifies the process of assigning privileges to individual users and facilitates application of least privilege. The model is also easily extended to incorporate new operations and objects when, and if they, are added to the system. The core RBAC definitions from the NIST model are shown in figure 2.9.

2.4.1 RBAC constraints

The generality and flexibility of RBAC make possible an almost endless array of constraints, and since RBAC is policy neutral, constraints play an important role in

allowing the RBAC model to enforce security policies. Simon and Zuko [68] presented the first discussions of constraints for RBAC in 1997 when they identified three kinds of constraints possible in RBAC systems:

- *Constraints on role membership* – overlap in member ship is constrained, usually to be null
- *Constraints on role activation* – legitimate users of a role may be prevented from assuming the role
- *Constraints on role use* – users who have assumed a role may be restricted in how that role is used.

Simon and Zuko limited their treatment of role based access control constraints to Separation of duty constraints (SoD), and much of the subsequent literature has also focused on separation of duty constraints in RBAC. Separation of duty is one of the eight design principles described by Saltz and Schroeder [69]. Separation of duty is a security policy concept based on division of responsibility and is a central component of Clark and Wilson's commercial security policy [70]. For example, consider the following three actions within a company: issuing a purchase order for an item, signing for the receipt of the item, and issuing a check to pay for the received item. Separation of duty would prevent the same person from carrying out all three actions, and thus potentially defrauding the company.

Simon and Zuko, in examining the literature in separation of duty identified two broad categories of separation of duty: strong exclusion or static separation of duty (SSoD), and weak exclusion or dynamic separation of duty (SSoD). In role based access control, two roles are strongly exclusive if no one person is ever allowed to perform both roles. Simon and Zuko pointed out that strong exclusion is simplistic and easily enforceable using only controls on role membership; however, it is often too ridged when applied to real world situations. Weak exclusion introduces the concept of time into

separation of duty, which is why it is usually referred to as dynamic separation of duty. Simon and Zuko define the flowing different type dynamic separation of duty with respect to role based access control:

- ***Simple Dynamic Separation of duty*** – restricted roles may be assigned to the same user, but a user may not use or assume both roles at the same time.
- ***Object-based separation of duty*** – restricted roles may be assigned to the same user and a user may assume or use multiple roles at the same time, but a user may not act on an object or target that the user has previously acted upon.
- ***Operational separation of duty*** – Restricted roles may be assigned to the same user as long as all the union of all permissions in the roles does not contain all the permissions in a given set of permission (which usually represent completing some business task).
- ***History-based separation of duty*** – extend object and operational separation of duty to allow for restricted roles to be assigned to the same user, and that user can carryout all permission in the union of the permissions for those roles but cannot carryout all the permissions in a given set (again modeling a business task) on the same target or object. Historical separation of duty constraints can be either order-dependent or order independent.

Another analysis of the types of possible constraints for role based access control systems was presented by Ahn [71]. Ahn and Sandhu [72] developed the RCL2000, an authorization constraint language for role based access control systems. In understanding the expressiveness of RCL2000, described in the next section, Ahn identified three classes of constraints that could be expressed in RCL2000.

- ***Prohibition constraints*** – Prevent an RBAC component from doing or being something
  Example: SOD (user cannot be a purchasing and payable manager)

- ***Obligation constraints*** – Force RBAC components to do or be something
  Example: Certain roles should be active in the same session or a user should have some combination of role assignments.

- ***Cardinality constraints*** – Limit the number of users, roles, sessions.
  Example: limit number of users assigned to a role or limit number of sessions a user is assigned.

As role based access control model increased in popularity, they began to be used in wider areas, opening up avenues for new types of constraints. Strembeck and Neuman [73] present a very comprehensive role based access control taxonomy. They define three dimensions, which they point out are not completely orthogonal, along which constraints can be categorized. The focus of their work is on only one type of constraint, specifically context constraints, which they point out are dynamic exogenous authorization constraints. Stembeck and Neuman's three dimensions are:

- ***Static constraints vs. dynamic constraints***
  Static constraints are constraints that can be evaluated directly at design time of an RBAC model. Dynamic constraints can only be checked at runtime according to actual values of specific attributes or with respect to characteristics of current sessions.

- ***Endogenous constraints vs. exogenous constraints***
  Endogenous constraints are constraints that are related to intrinsic properties of an RBAC model and inherently affect the structure and construction of a concrete instance of an RBAC model. Exogenous constraints apply to attributes that are not apart of the core RBAC model.

- ***Authorization constraints vs. assignment constraints***
  Constraints that place additional controls on access control decisions, such that a subject may possess the appropriate permission but can be prevented access by one or more authorization constraints. Assignment constraints are constraints that control the assignment of permissions and roles.

2.4.2 Specifying constraints on RBAC models

The previous section presented a variety of categorical definitions for role based access control constraints. As pointed out by Jaeger [74] constraints are becoming increasingly important in role based access control systems since they provide a means to ensure that role specification actually matches desired access control requirements. However, to take advantage of role based access control constraints, a role based access control model must be capable of expressing constraints, and accurately interpreting them. This section presents a number of role based access control models that include the ability to express constraints. The results are summarized in table 2.5, which shows the different types of constraints a given model can express.

Giuri and Iglio [75] present one of the early role based access control models that included constraints. The model is an extension to the named set of protection domains (NSPD) model [76] in which a role is defined NSPD. A NSPD specifies a collection of possible sets of privileges:

(2.1) $\{\{priv_{1,1}, \ldots, priv_{1,i}\}, \ldots, \{priv_{n,1}, \ldots, priv_{n,j}\}\} = \{Pd_1, \ldots, Pd_n\}$

Only one protection domain can be active at a given time. Girui and Iglio extended NSPD to include constraints:

(2.2) $\{<c_1, \{priv_{1,i}, \ldots, priv_{1,j}\}>, \ldots, <c_n, \{priv_{n,1}, \ldots, priv_{n,j}\}>\} =$
$\{<c_1,Pd_1>, \ldots <c_n,Pd_n>\}$

where c is an expression in first order logic, or possibly an SQL query. A constrained protection domain is activated only if the corresponding constraint is satisfied. While the model includes constraints, Guiri and Iglio's model is limited to expressing exogenous dynamic role activation constraints.

As mentioned previously, the variety of different role based access control models developed during early research on role based access control made comparing different models very difficult. The NIST standard role based access control model, developed in 2000, was an attempt to address this issue. The NIST RBAC model includes two important categories of extensions: role hierarchies and constraints. In role hierarchies a hierarchical relation on roles (i.e. manager – supervisor – clerk) is defined and more senior roles inherit the permissions associated with junior roles. Of interest here is constrained RBAC model described in the NIST standard. In the NIST model, there are two types of constraints: static separation of duty (SSD) and dynamic separation of duty (DSD) each specified in the model as a collection of pairs of the from {rs,t} where rs is a set of roles and t is an integer >= 2. SSD is formally defined as:

$$(2.3)\quad \forall (rs,n) \in SSD, \forall t \subseteq rs : |t| \geq n \Rightarrow \bigcap_{r \in t} assigned\_users(r) = 0$$

And DSD is formally defined as:

$$(2.4)\quad \begin{aligned} &\forall rs \in 2^{ROLES}, n \in \mathrm{N}, (rs,n) \in DSD \Rightarrow n \geq 2 \wedge |rs| \geq n, and \\ &\forall s \in SESSIONS, \forall rs \in 2^{ROLES}, \forall role\_subset \in 2^{ROLES}, \forall n \in \mathrm{N}, (rs,n) \in DSD, \\ &role\_subset \subseteq rs, role\_subset \subseteq session\_roles(s) \Rightarrow |role\_subset| < n. \end{aligned}$$

Table 2.5. Types of constraint supported in different RBAC models.

Citation	Obligation	Prohibition	Cardinality	Dynamic	Static	Historical	endogenous	exogenous	Authorization	Assignment
Giuri and iglio 1997	yes			Yes				Yes	yes	NO
Ahn 2000 RCL2000	Yes	Yes	No							
NIST, 2000	No	Yes	Yes	Yes						
Shin, 2003		Yes	No	Yes	No	Yes	Yes	No	Yes	No
Crampton, 2003	No	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes
GTRBAC 2005	Y?	Y	N	Y						
Li et.al, 2005 Smer constraints	No	Y	N	N	Y	N	Y	N	Y	N
Strembeck and Neumann, 2004	Y	N	N	Y	N	N	N	Y	Y	N

In the NIST model the SSD constraint limits the UA relation by defining sets of roles such that a user can only be assigned some proper subset of the roles. A DSD constraint is similar except that they restrict the active roles of a user's session rather than the assigned roles. Figure 2.10 shows the relationship of SSD and DSD to the standard RBAC model. The constraints allow RBAC to express separation of duty or conflict of interest relations [77]. The NIST RBAC reference model constraints are thus limited exclusively to role assignment and role activation, and are endogenous, but include both static and dynamic constraints.

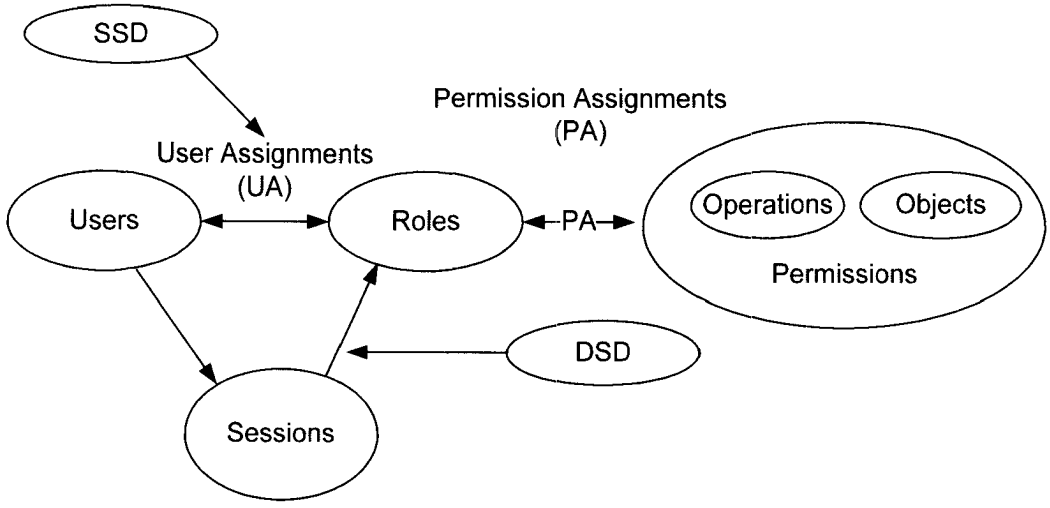


Figure 2.10. Core RBAC with SSD and DSD constraints defined in [67].

The use of static and dynamic separation of duty (SSoD & DSoD) however, is somewhat misleading as pointed out by Li, Bizri, and Tripunitara [78]. They point out that separation of duty (SSoD) is an objective not a mechanism. The mechanism, which is what is actually described in the NIST model, are static mutually exclusive roles, or *smer* for short. Li et al. define Static Separation of Duty (SSoD) as a set of m permissions and an integer $k < m$ indicating that a there should not exist a set of less than k users that together have all m permissions. This differs from a SMER which is a set of

m roles and integer t < m forbidding any user from being a member of t or more roles in $\{r_1 \ldots r_m\}$. Li et al go on to point out that there then exists a verification problem: does a set of *smer* constraints achieve a given SSoD goal, and a generation problem: how do we generate a set of *smer* constraints that adequately enforce a SSoD policy? Li and colleagues demonstrated that directly enforcing SSoD polices is coNP complete but that enforcing *smer* constraints is efficient, and went on to develop an algorithm to generate a set of *smer* constraints given an SSoD policy.

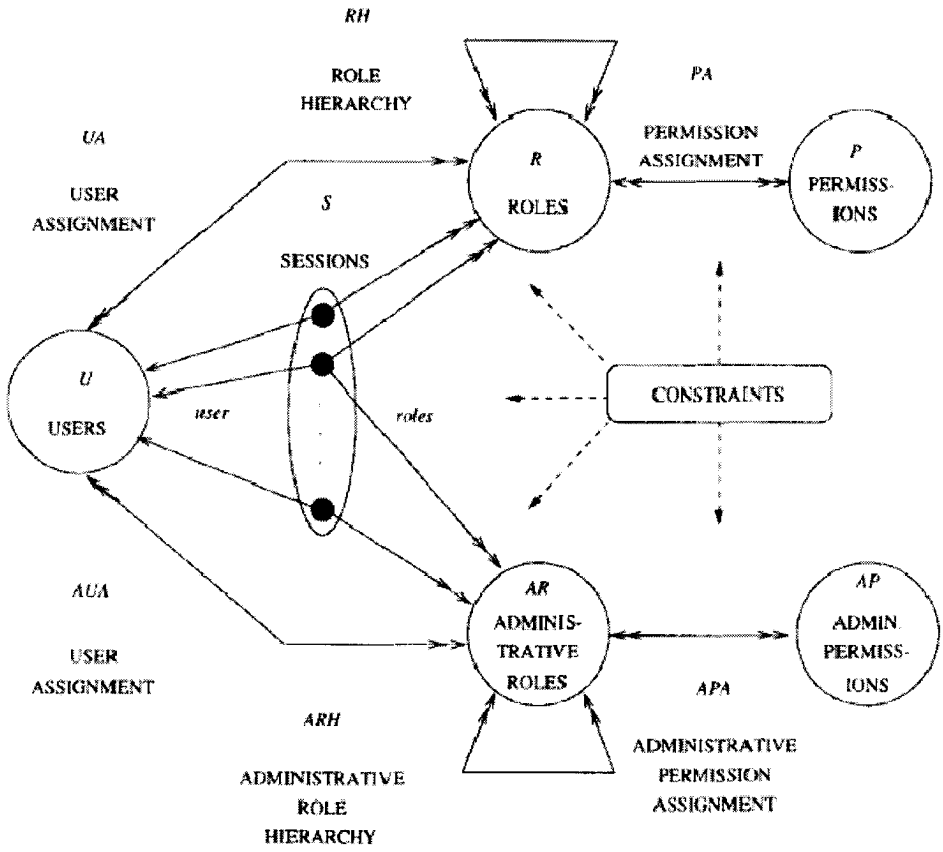
The constraint specification described by Ahn and Sandhu [72] called RCL2000 is based on the ARBAC 97 [79] model. ARBAC 97 is similar to the NIST reference model except that it includes administrative permissions and administrative roles. A summary of ARBAC 97 is shown in figure 2.11. Rather than leave administration of the RBAC to an external "trusted" security administrator, theARBAC97 model includes administrative elements in the model, allowing administration of the policy to be distributed among multiple roles and multiple users. RCL2000, developed by Ahn and Sandhu and based on previous work by Chen and Sandhu [80], provides a constraint specification language and extends ARBAC97 with the following elements:

- CR: A collection of conflicting role sets {cr1 ... crn} cri = {r1..ri} $\subseteq$ Roles
- CP a collection of conflicting permissions sets {cp1..cpn} cpi ={p1..pi} $\subseteq$ Perms
- CU a collection of conflicting user sets {cu1 .. cun} cui = {u1..un} $\subseteq$ Users
- Two non-deterministic operators
    - $OE(X) = x_i$ where $x_i \in X$ (called the oneeach operator)
    - $AO(X) = X - \{OE(X)\}$ (called the allother operator)

The model assumes that initially a policy administrator or developer defines the sets CR, CP and CU. Once each of these sets is defined then RCL2000 expressions can be used to define policy based constraints. The syntax of an RCL 2000 expression is given in figure 2.12. An RCL expression can best be understood through an example. The formula (2.5)

is an RCL expression, and its interpretation would be: no user can be assigned to two conflicting roles.

(2.5) $|roles(OE(U)) \cap OE(CR)| \leq 1$

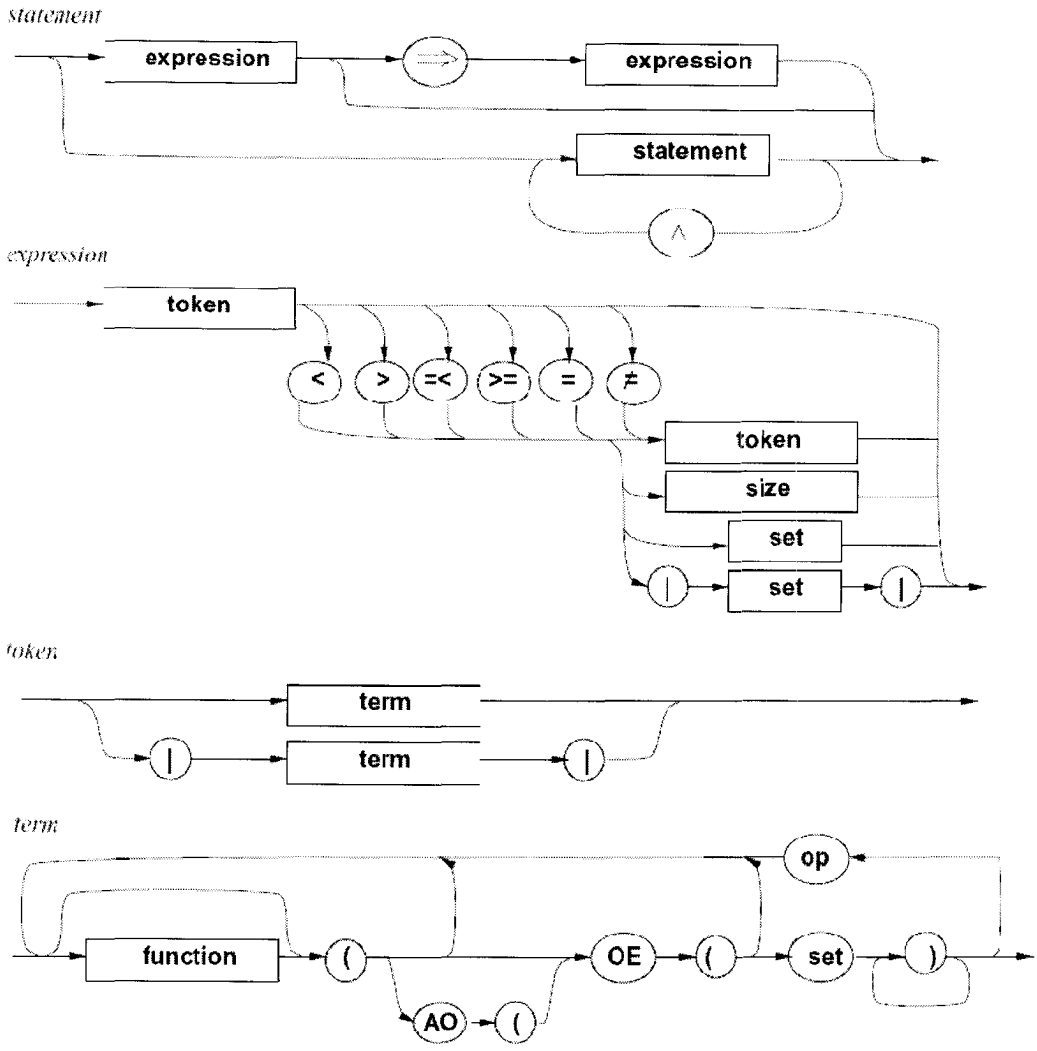


—U, a set of users; R and AR, disjoint sets of (regular) roles and administrative roles; P and AP, disjoint sets of (regular) permissions and administrative permissions; S, a set of sessions
—$UA \subseteq U \times R$, user to role assignment relation
$AUA \subseteq U \times AR$, user to administrative role assignment relation
—$PA \subseteq P \times R$, permission to role assignment relation
$APA \subseteq AP \times AR$, permission to administrative role assignment relation
—$RH \subseteq R \times R$, partially ordered role hierarchy
$ARH \subseteq AR \times AR$, partially ordered administrative role hierarchy
(both hierarchies are written as $\geq$ in infix notation)
—$user : S \rightarrow U$, maps each session to a single user (which does not change)
$roles : S \rightarrow 2^{R \cup AR}$ maps each session s_i to a set $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA \cup AUA]\}$ (which can change with time)
session s_i has permissions $\cup_{r \in roles(s_i)}\{p \mid (\exists r'' \leq r)[(p, r'') \in PA \cup APA]\}$
—there is a collection of constraints stipulating which values of the various components enumerated above are allowed or forbidden

Figure 2.11. Summary of the ARBAC 97 Model [79].

RCL2000 is a specification language. Syntax and semantics are defined along with a means of converting between RCL2000 and a restricted form of first order predicate logic. A soundness and completeness proof for the conversion is given. Finally the expressiveness of RCL2000 is shown to include lattice-based access controls as well as traditional and new separation of duty constraints, but no discussion of an enforcement mechanism is given. RCL2000 can express prohibition, obligation, and cardinality constrains, static constraints, some types of dynamic constraints, and is strictly endogenous.

A very different approach to role based access control with constraints was proposed by Wook et al. [81]. Wook's model, extended role based access control with procedural constraints, is intended to be used in trusted operating systems. Motivation for the model comes from preventing classic timing attacks that relink */etc/passwd* to give an attacker write permission to */etc/passwd.* Wook adds negative procedural constraints to the classic RBAC model to prohibit certain sequences of ordinary actions, such as relinking */etc/passwd.* The procedural constraints operate on behaviors, and these behaviors are interposed between the traditional role permission assignment. Colored Petri nets are used to model behaviors, and detect when a behavior has become malicious. This model provides only dynamic historical constraints, but can express order dependent constraints.



op ::=∈| ∩ | ∪

size ::= ϕ | 1 | ... | *N*

set ::= `U` | `R` | `OP` | `OBJ` | `P` | `S` | `CR` | `CP` | `CU`

function ::= `user` | `roles` | `roles*` | `sessions` | `permissions` | `permissions*` |
`operations` | `object` | `OE` | `AO`

Figure 2.12. RCL 2000 syntax.

One of the more recent and elegant role based access control constraint models was developed by Jason Crampton [82], which extends the classic role based access control model described in section 2.5.1. Crampton points out that constraints informally define bad sets. He then invents a formal specification scheme to allow formal

specification of constraints in the form of a triplet triple (s, c, x) where s is the scope, c is the constraint set, and x is a temporal context either {static, dynamic, historical}. The scope and constraint sets are a subsets of U, R, and P. A constraint defines a family of sets. For example, 2.6 states that neither user u_1 nor user u_2 can be assigned both r_1 and r_2.

(2.6) $(\{r_1,r_2\},\{u_1,u_2\},s)$

This specification scheme can express both static, dynamic and historical constraints on role and permissions and is strictly endogenous. Crampton also suggests the use of black lists to enforce historical constraints, though this proves difficult for cardinalities greater than two.

In addition to the taxonomy for constraints, Stembeck and Neuman [73;83] also developed a model to express and enforce context based exogenous authorization constraints. Their model begins with the classic model and adds the elements shown in figure 2.13.

Strembeck and Neuman's model is specifically dynamic and exogenous. When the check permission action is performed on a requested action, the enforcement point first checks to see if the subject is assigned to a role that has the appropriate permission. If the subject is assigned to such a role, then, before validating the request, any context constraints on that permission are evaluated. If a context constraint condition evaluates to false, then the requested operation is denied. If every context constraint evaluates to true, then the operation is allowed. Possible conditions suggested by Strembeck and Neuman include the IP address of the subject must be a certain value, the time or date must fall in a certain range, or even specific conditions about a user, their age or gender

for example. The model only supports dynamic exogenous authorization constraints, though it is easy to see that the NIST SSoD or DSoD constraints could be included to add support for other types of constraints. Strembeck and Neuman also present an execution model and approach for discovering context constraints in a specific environment.

—ATTS, the set of context attributes (e.g., local_time, local_IP_address, subject_name, subject_age).
—DOMAINS, the set of available domains (e.g., boolean, date, integer, real, string).
—CONSTANTS = $\{x \mid x$ is a constant value $\wedge\ domain(x) \in$ DOMAINS$\}$.
—OPERANDS = ATTS $\cup$ CONSTANTS
—OPERATORS, the set of available (comparison) operators, for example, infix operators as $=, \geq, >, <, \leq, \neq$.
—$domain(oprtr :$ OPERATORS$) \rightarrow \{d \subseteq$ DOMAINS$\}$, a function to determine the set of domains an operator is specified for.
—$domain(oprnd :$ OPERANDS$) \rightarrow \{d \in$ DOMAINS$\}$, a function to determine the type of an operand.
CONDITIONS $= 2^{\text{OPERANDS} \times \text{OPERATORS}}, \forall c \subset$ CONDITIONS $: c \mapsto \{(oprnd_1, \ldots, oprnd_x, oprtr) | oprnd_1, \ldots, oprnd_x \in$ OPERANDS$, oprtr \in$ OPERATORS$\} \wedge \{domain(oprnd_1) \cup \cdots \cup domain(oprnd_x) \subseteq domain(oprtr)\}$.
—CC $= 2^{\text{CONDITIONS}}$, the set of context constraints.
—$conditions(cc :$ CC$) \rightarrow \{cond \subseteq$ CONDITIONS$\}$, a function to determine the conditions linked to a certain context constraint.
—PCL $\subseteq$ PRMS $\times$ CC, a many-to-many permission to context constraint linkage relation.
—$linked\_ccs(p :$ PRMS$) \rightarrow \{constraints \subseteq$ CC$\}$, the linkage of a permission p to a set of context *constraints*. Formally: $linked\_ccs(p) = \{c \in$ CC $\mid (p, c) \in$ PCL$\}$

Figure 2.13. Context constraints as defined by Strembeck and Nueman [73]

Jaeger and Tiswell [74] developed a graphical based constraint specification scheme. The scheme is actually broader than role based access control, and was aimed at expressing all the different types of separation of duty. The model is very expressive and shared some similarities to RCL2000. The model is based on DTAC not RBAC, but they are similar enough (basically DTAC is RBAC with object type and some other extensions) to make types of constraints possible. The model is expressed as a graph, G(X,Y). X is a set of nodes, each node is a set, but of possibly different types. Y is set

of edges representing relationships between sets. Constraints are defined as binary relationships between sets. Figure 2.14 shows an example of a constraint specification using Jaeger and Tiswell's notation. In the figure p1 and p2 are arbitrary permissions and the double headed arrow marked with $\perp_T$ indicating that permission p1 may not be assigned any authorization type to which permission p2 is assigned, and vice versa.

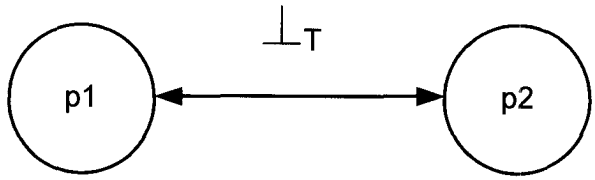


Figure 2.14. A graphical constraint specification in [74].

2.5 Operating system security, reliability and reduced kernels

The heart of an operating system is the kernel. The kernel is the part of the operating system that executes in the processor's privilege mode and provides the lowest level of abstraction between the physical components and the rest of the system. Thus the kernel plays a key role in the security of a system since it mediates all or almost all access to the computer systems physical resources. In today's multiprogrammed operating systems, the kernel is responsible for providing a means for principals (users or processes) to share systems resources by providing a common interface to those resources and controlling access to the shared resources. The security and reliability of an operating system are determined by how well, or how consistently, the operating system provides that protection. When the kernel fails to consistently enforce the appropriate protection, users experience system failures and/or security violations. The trustworthiness, or the degree that a system can be trusted, reflects the confidence that the operating system, usually the kernel, provides appropriate and adequate protection. The

frequency with which modern commercial operating systems, such as Linux and Windows, fail to adequately provide protection is all too familiar to users today.

One of the essential ideas in the construction of secure or trusted systems has been the security kernel. The security kernel is based on the concept of a reference monitor and a reference validation mechanism as discussed by Bishop in [77]. A reference monitor is an access control concept of an abstract machine that mediates all access to objects by subjects. A reference validation mechanism is an implementation of a reference monitor that can be proven to be tamperproof, always invoked, and small enough to be subject to verification. A security kernel is then defined to be the combination of hardware and software that implements a reference monitor. Later the idea of a security kernel was extended to trusted computing base (TCB), which is the collection of all protection systems responsible for enforcing a security policy.

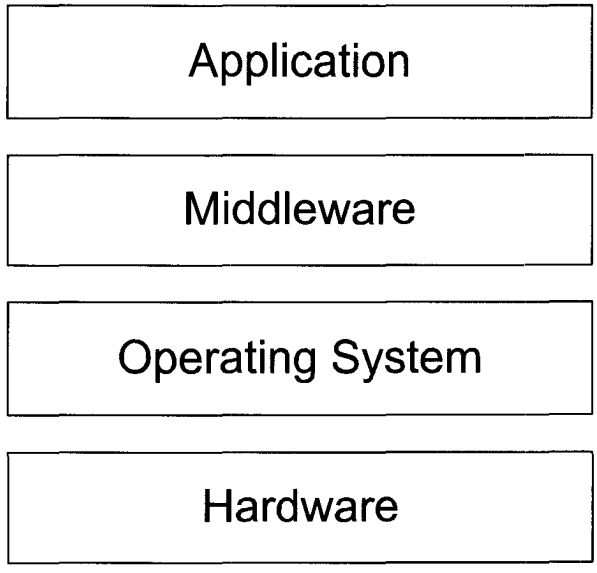


Figure 2.15. Computer architectural layers.

A major contributing factor to the low level of reliability and security provided by today's commercial operating systems is the fact that they use a monolithic kernel design

that has led to large kernels with poor fault isolation [84]. In monolithic kernels all of the core operating system functionality is implemented in the kernel. This functionality includes: memory management, file systems, access control, network stacks, device drivers, and interrupt handling. Therefore the TCB, the entire kernel, is very large and thus difficult to analyze. A typical monolithic kernel design is shown in figure 2.16, with the TCB shaded in gray.

In order to support the increase in variety of hardware available, the size of the kernel in commercial operating systems has become surprisingly large and is continuing to grow [85]. There are now about 4.1 million lines of code (LOC) that make up the Linux kernel, and Vista is said to have 20 million LOC [85]. Much of that code runs in privilege mode of the processor, allowing it unrestricted access to system resources, including memory and IO ports. Therefore, software flaws in the kernel code can potentially do a great deal of damage since these processes are not subject to system protection mechanisms. Unfortunately software errors or defects cannot be avoided. Due to the size and complexity of the kernel, finding all the defects before deployment is simply not feasible, and finding them all, in any given amount of time is unlikely. This led to a penetrate and patch approach to operating systems security and reliability. As pointed out by Loscocco and colleagues [86] the assumption that adequate security can be achieved at the application layer is seriously flawed. Without a secure operating system, application layer security mechanism cannot succeed.

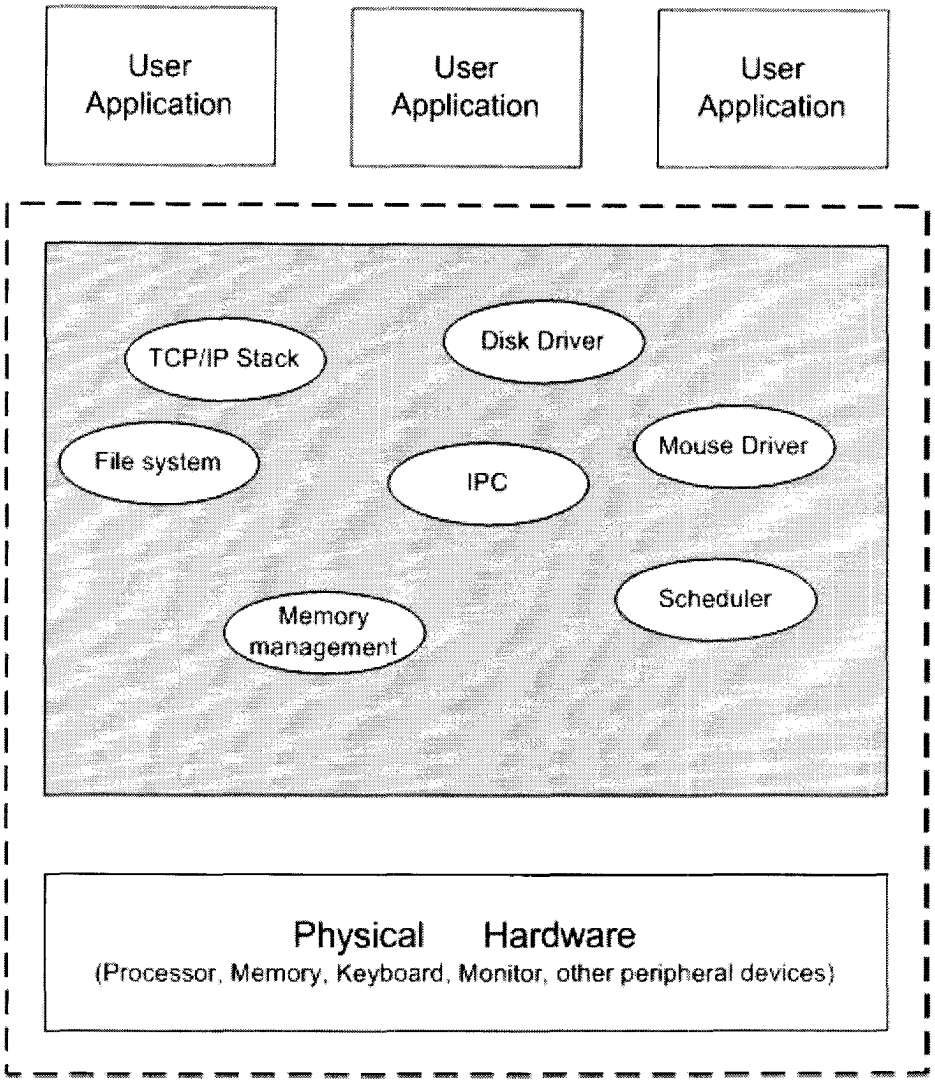


Figure 2.16. Monolithic kernel design.

The size and complexity of commercial operating systems is only continuing to increase. The problem is further exacerbated by the fact that within a monolithic kernel there is no fault isolation. If some piece of code in the kernel has a defect, then exploiting that defect can corrupt the entire kernel (reliability) or give malicious code the opportunity to bypass the normal access control measures (security vulnerability) that protect systems objects and other principles.

2.5.1 Minimal kernels

A small TCB is more amendable to formal analysis, making possible a much more trustworthy TCB. To achieve this a minimal kernel is needed. One approach that is receiving considerable attention recently is the microkernel designs. A microkernel [87], is a minimal kernel that implements only those services that cannot be implemented in user space. There are three minimal requirements for microkernels described by Liedtke [87]: *address spaces*, *inter-process communication*, and *unique identifiers*.

Microkernels have actually been around in some form for quite a while. The first system that could be considered a microkernel was Brinch Hansen's Nucleus [88]. Hensen's Nucleus supplied only primitives for process control and inter-process communication, with the operating system policy and strategies implemented outside the kernel. Hydra [89] extended the Nucleus work, and was instrumental in separating policy (in user land) from mechanism (in the kernel). Following on Nucleus was Mach [90] developed at Carnegie Mellon University. It was the Mach team that coined the phrase microkernel. Interestingly enough, the Mach kernel, while calling itself a microkernel, contained nearly 150 thousand lines of code and had 200 systems calls. There were many contemporaries of the Mach kernel such as NextStep [91] (which became Mac OS X), University of Utah's Mach4 [92], and IBM's Workplace OS [93]. In the 1980's microkernels received a great deal of attention and focus. However poor performance characteristics plagued developed systems. The reasons for this, and some alternatives, are discussed in the following paragraphs.

Central to the microkernel architecture, shown in figure 2.17, is the notion of user land operating services that replace many of the operating systems services that are

included in the monolithic kernel. Moving these services to user land provides better fault isolation and prevents errors in a specific service from allowing complete compromise of the system. To allow these services to communicate, the microkernel has to provide inter processes communication (IPC) functionality to all the supported tasks or threads. Microkernels of that time exhibited IPC costs of about 100 microseconds [85] and in 1994 Chen and Bershad concluded that performance problems were "inherent in the OS structure" [94]. Many then concluded that the microkernel approach could not meet the performance demands and interest in them waned.

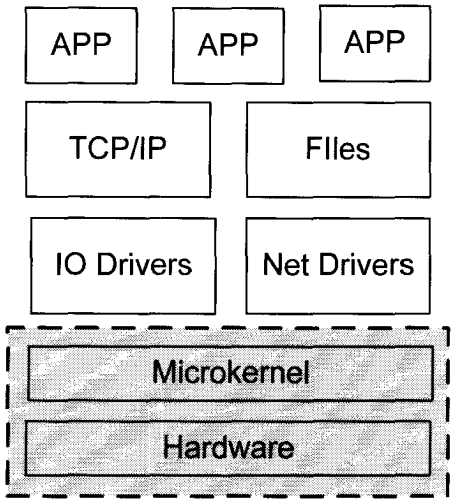


Figure 2.17. The basic microkernel design.

Around 1994 John Ledieke and others began analyzing microkernels and found that microkernel IPC could be made fast through strict adherence to the minimal kernel goal and optimization [95]. Leidke developed the L4 kernel to demonstrate the performance improvement, and demonstrated an order of magnitude improvement over the Mach microkernel. The bad performance reputation that microkernels had gained prevented Leidke's work from receiving attention at the time. But today microkernels are receiving renewed attention due in part to the popularity of virtualization and hypervisors

which have a lot in common with microkernels. One of the areas of renewed interest is the Rushby's idea of a separation kernel, which is explained in the next section.

2.5.2 Separation kernel and the MILS architecture

Recall that the TCB concept grew out of the reference monitor first developed in the 1970's. There has been substantial work to implement reference monitors but problems have been encountered in the application and implementation of these principles [86;96;97]. Rushby argues that the problems encountered in constructing and verifying security kernels is that they attempt to impose a single security policy over the entire system. Instead, Rushby proposes leveraging the inherent security benefits of physically distributed systems. In physically distributed systems, such as a network printer, network storage, and a PC connected to the network, security is achieved through a combination of physical separation of individual components, and mediation of trusted functions within some components [96;98]. To support a distributed system on single processor, Rushby proposed the idea of a separation kernel.

The Multiple Independent Levels of Security or (MILS) architecture was developed to provide a high-assurance and high-performance computing architecture with the ability to enforce strict security and separation policies on data and processes residing on a single processor [99]. The architecture was designed to make possible formal verification of application reference monitors and therefore realize high assurance of security mechanisms. The MILS concept originated with Rushby's work and is based on his separation kernel. The current MILS architecture is pictured in figure 2.18. A separation kernel has four security requirements [97]:

- Data Isolation: Information is accessible only by that partition, and private data remains private,
- Control of Information Flow: Information flow from one partition to another is from an authenticated source to authenticated recipients; the source of the information is authenticated to the recipient, and information goes only where intended,
- Periods Processing: The microprocessor and any networking equipment cannot be used as a covert channel to leak information to listening third parties,
- Fault isolation: Damage is limited by preventing a failure in one partition from cascading to any other partition. Failures are detected, contained, and recovered locally.

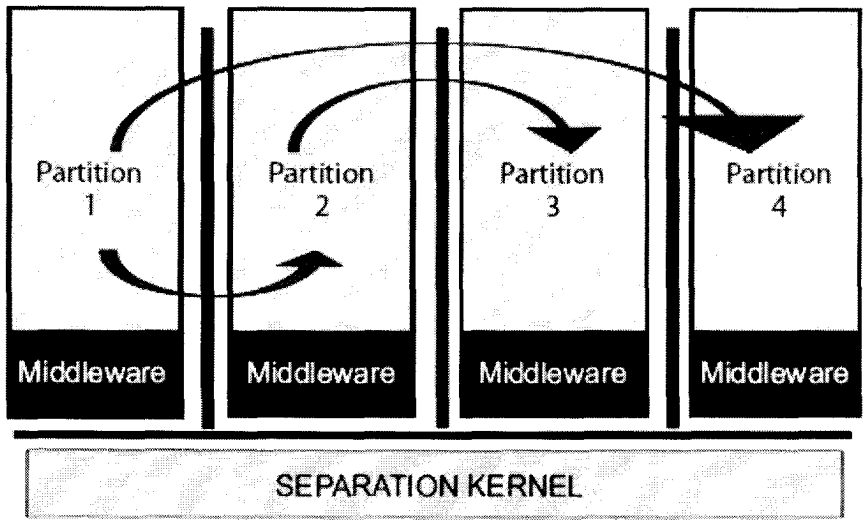


Figure 2.18. The MILS architecture [100].

The security advantages of the MILS architecture is that it provides both process separation and functional separation. Process separation strictly enforces the flow of information between user processes, such that information flows only when explicitly permitted. Through functional separation, MILS moves security functions out of the kernel and into modular external components. These components can enforce specific security policies and then can be layered together to provide an overall system security policy. Because the MILS architecture provides data isolation and information flow control, an implementer can create an application-level reference monitor in a user partition that is NEAT (non-bypassable, evaluatable, always invoked, and tamperproof

[100]). An additional advantage of these independent modules is that they are small enough, and simple enough, to be evaluated using formal techniques; thus making it possible for the entire system to achieve a high level of assurance. An example implementation of a mediator (or reference monitor) using physical separation and the MILS architecture is presented by Hanebatte and colleagues [99].

CHAPTER III

SECURITY HARDENING RTUS

Chapter two presented the security threats and vulnerabilities that currently face SCADA systems in general. The traditional IT security approach of soft on the outside (peripheral systems) and hard in the middle (servers) is not appropriate for SCADA systems. In SCADA installations the peripheral devices, such as RTU, must be security hardened as well. The focus of this dissertation is to identify and develop hardening techniques for RTUs and to develop a security hardened RTU. As discussed in the previous chapter, in the past, these devices faced primarily physical threats, but today they are increasingly network enabled and network accessible. Security hardening these devices is a major challenge facing the development of secure SCADA systems. Two security hardening approaches are explored in this dissertation, an RTU role based access control model and a reduced kernel OS. Previous work on both role based access control constraints and minimal kernels for operating systems was presented in chapter two. This chapter introduces the architecture for a security hardened RTU. The RTU role based access control model is presented in detail in chapter four and the minimal kernels for RTU are presented in chapter five.

Before considering specific RTU threats it is important to define, from a security perspective, the security perimeter of an RTU. This approach parallels the definition of a physical security perimeter that is a standard approach in securing physical places. SCADA systems are large distributed systems, and in developing a layered approach to

security for them it is important to identify security boundaries for different components. For field devices, the security boundary, or electronic perimeter, is defined to be the point at which the device makes contact with the SCADA network. For example, if the RTU connects to the SCADA network using Ethernet, then the electronic perimeter is the Ethernet controller card. It is important to establish such a perimeter; if the perimeter were too encompassing, the RTU's security perimeter would include components over which it has no control.

3.1 RTU security vulnerabilities

Vulnerabilities in RTUs can occur at many different layers. The highest and most abstract layer is the protocol layer. Protocols are abstract descriptions, and must be implemented, typically in software. Below the protocol layer is the software application layer which will implement interfaces to the SCADA network, particularly SCADA protocols, but this also applies to other protocols that might be used now or in the future. It is rare today that software applications are written to run at the hardware level. Instead software usually makes use of libraries, and other applications to achieve its goals. This creates yet another layer of shared software libraries and binaries, which is often referred to as middleware. Below the middleware layer is the operating system kernel, the lowest level of abstraction between the hardware and the software; finally, there is the actual hardware. Figure 3.1 shows these architectural layers.

As mentioned in chapter two, initial SCADA protocols did not include security features, which resulted in vulnerabilities to message modification, spoofing, and sniffing attacks. The protocol vulnerabilities are really outside the RTU security perimeter, at least in their specification, but it is important to mention them, and keep them in mind in

considering lower layers. One reason for keeping these in mind is the principle of easiest penetration [101]. If an RTU supports an insecure SCADA protocol that can easily be attacked and used to control or damage the connected physical processes, it will be impossible for a lower level prevention mechanism to protect the RTU since it has no way of differentiating between authentic and un-authentic SCADA communications. Two excellent solutions to address the shortcomings in SCADA protocols have been presented in the literature review in chapter two [29;52]. Both the AGA's cryptographic solution, and Patel's authentication octets and challenge response approach adequately address the vulnerabilities in SCADA protocols.

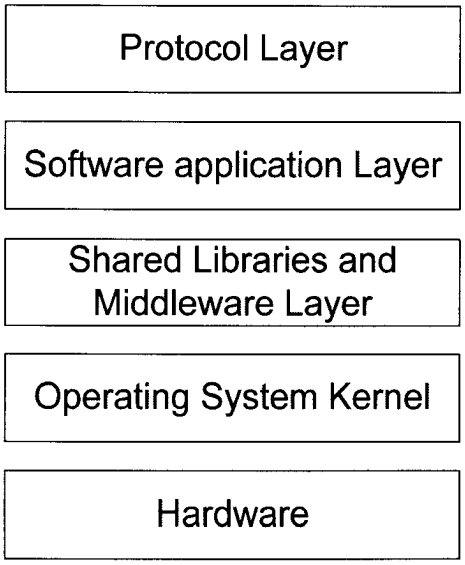


Figure 3.1. RTU architectural layers.

Unfortunately, though providing for authentication in the protocol is a significant improvement, there are still many additional RTU threats and vulnerabilities. One of the more significant threats is that of insider attacks. In the 2000 CSI computer crime survey, cited by [6], insider attacks represented 71% of security breaches. Much of the perpetrated computer crime is the result of insiders; they represent a big threat since they have knowledge of systems and networks, and valid authentication credentials. For

example, the recent hacking of a waste management system in Australia was carried out by an insider [5]. An insider might be a disgruntled employee or ex-employee as was the case in Australia. With respect to RTU security, an insider could have knowledge of the network location of RTUs, such as their phone number or an IP address, and if the RTU is protected by some form of authentication then a user might have knowledge of the password or other form of identification. Not all insiders are necessarily malicious either. It is possible that a legitimate user might inadvertently cause damage by changing the value of a set point they were not supposed to change or accidentally issuing a control command.

Another major vulnerability for RTUs is software vulnerabilities. A protocol may be secure, but its implementation may contain flaws. These flaws can be exploited by an attacker to circumvent or by-pass the security provided by the protocol. Software vulnerabilities can also be found in other COTS software components that might be included in an RTU. For example, the trends discussed in chapter two, particularly the increase use of Ethernet, are leading vendors to include pervasive commercial components such as such as FTP servers, Web servers, or a remote access server. These COTS systems may also contain vulnerabilities; for example, the well known Wu-ftp exploit [102], and can be exploited just as easily when they are deployed on RTUs as they can when deployed on typical desktop systems. When vendors adopt commercial operating systems, many of these services such as telnet, FTP and others come preinstalled and possibly activated. These commercial pieces of software may also have flaws, perhaps even well known ones, which could be exploited.

There may also be flaws in shared libraries or other pieces of common code used by RTU applications. These shared resources can also have exploitable vulnerabilities. Many RTU applications are written in C or C++ which have limited buffer overflow prevention capabilities, and make use of vulnerable C libraries.

As discussed in chapter two, the operating system kernel is responsible for providing protection and allowing sharing within a computer system. Operating systems themselves can have flaws. Monolithic kernel commercial systems tend to have a lot of flaws, and hence a lot of software vulnerabilities. For example, it is well known that commercial operating systems like Windows and Linux contain flaws and require the application of patches on a regular basis. Recent estimates put the number of flaws in the Linux kernel at 15000 and for Windows XP nearly 30000 [103]. The penetrate and patch approach that is familiar to commercial operating systems is often not applicable to RTUs It is difficult to regularly apply patches to RTUs simply because they are remote and because they cannot endure the risk that a patch breaks the operation of some critical RTU components.

Once an attacker has managed to gain access to the RTU, there are the obvious threats that the attacker will issue control commands to connected actuators and disrupt operation of the SCADA system. Other threats include deleting or modifying stored data to hide evidence of the attack and replacing or modifying application code to corrupt system integrity. In addition, a privilege escalation threat also exists, where an attacker, having gained some level of access, attempts to increase their rights by attacking the access control configuration.

3.2 A security hardened RTU

To address these vulnerabilities, this dissertation proposes two security hardening techniques for RTUs and presents these in a security hardened RTU. The security hardened RTU employs three security enhancements:

1. The use of a security enhanced SCADA protocol
2. Fine grained access control
3. A reduced kernel OS

The secure SCADA protocol used is that developed by Patel [29] and is used to provide RTU authentication of users. The focus of this dissertation is the access controls and reduced kernel OS. Section 3.2.1 briefly describes the security enhancement proposed by Patel [29]. Section 3.2.2 introduces the RTU access control model and section 3.2.3 introduces two reduced kernel approaches for a reduced RTU OS kernel.

3.2.1 Secure SCADA protocol

The focus of this dissertation is not on security in the protocol layer or secure SCADA protocols. However the security hardened RTU must have a secure protocol, otherwise other security measures can too easily be circumvented. A new security model for SCADA communications was presented by Patel in [29]. This enhancement uses a challenge response approach to allow either party to perform spontaneous sender authentication and message integrity verification of the most recently received message. Authentication using challenge response allows periodic verification of the identity of the communicating party and the integrity of the most recent message. The challenge response mechanism requires that all parties possess a pre-shared secret. Either of the

devices (a master or an outstation) can initiate the challenge. The steps are as the following:

1. After the link is established, the authenticator (a master or a field device) sends a random "challenge" message to the other party (a field device or a master).

2. The other party sends a response message that includes an HMAC [104] of the challenge message. The challenge message includes a pre-shared secret that assures only a valid device can calculate the correct hash.

3. The challenger checks the response against its own calculation of the expected hash value. If the values match, the normal operation proceeds; otherwise the connection gets terminated.

4. At random intervals, the authenticator sends a new challenge to the other party, and repeats steps 1 to 3.

To enhance security, devices issue challenges at the following times:

- initially, to prevent any communications from proceeding without prior authentication,
- periodically and randomly, to protect against man-in-the-middle attacks,
- and before carrying out specific critical device operations. (The challenger issues the challenge immediately after receiving the critical operation, and before taking any action on it.)

To protect against replay attacks, the challenge message contains a nonce value that changes randomly each time a challenge is issued. The challenger, in the challenge message, specifies the cryptographic algorithm to be used in calculating the HMAC when building the response.

The responder performs the cryptographic algorithm specified in the challenge message to produce a response. The following information is included in the computation:

- Address information from the SCADA protocol (in order to authenticate the responder as a valid application layer user)
- The challenge data (to protect against replay attacks)
- The requested operation (if the challenger is protecting a specific critical operation)
- A shared secret known to both outstation and master

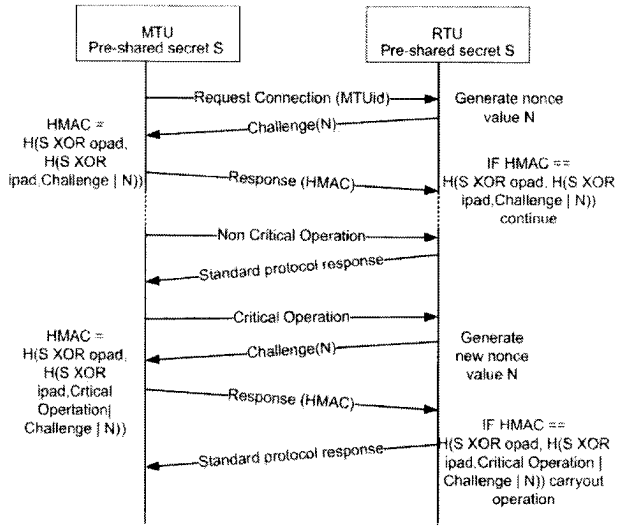


Figure 3.2. Challenge response authentication [105].

3.2.2 RTU role based access control

While it is impossible to eliminate insider threats and the potential failure of authentication schemes, like the one presented in section 3.2.1, it is possible to greatly limit the threat and constrain the potential damage of these attacks using access control as opposed to access restriction. As has been discussed previously, many SCADA systems employ neither, but those that do favor access restriction, as is fits with the physical protection model that is more familiar to industry practitioners. In access restriction, there is a many-to-one mapping of users to an identity, as opposed to access control where there is a one-to-one mapping of users to identity. In access restriction, any authenticated user is allowed to carry out any valid operation since the system cannot distinguish between users. Access control provides improved security by allowing users access to only certain permissions.

Another vulnerability that can be mediated by access control is the trend toward open systems and interchangeable SCADA components. Using well defined open protocols, such as DNP3 and Modbus, led to the case where many RTU devices, share a

common stack implementation. For example, Triangle Microworks has a DNP3 stack which RTU vendors can purchase and integrate into their devices. While there are advantages to this, the disadvantage is that a vendor's RTU may support functionality of which they are not aware. For example, a purchased DNP3 or Modbus stack will likely include all of the functionality described in the protocol specification; a vendor, not intimately familiar with the protocol, might only pay attention to those functional elements needed by their device. Also a SCADA operator or implementer may want to absolutely limit what a particular protocol can do. For example, suppose they need to provide access to a business partner to parts of the RTU data. Providing that access through a standard protocol, such as DNP3 or Modbus, the SCADA operator cannot easily limit the capabilities provided to a business partner. The SCADA operator may even be unaware that certain capabilities are exposed.

A primary contribution of this dissertation is an access control model for RTUs. In developing the RTU access control model, different access control models including the access control matrix model [64], the HRU model [106], lattice models [65;107;108] and role based access control models [67;109] were considered. While all of these models have strengths and weaknesses, RBAC was chosen for modeling because of its flexibility and expressiveness. RBAC is policy neutral and can support fine grained access control enforcement. Another advantage of RBAC is that it more accurately captures the collection and assignment of permissions to users. It is much more logical to think of an RTU operation, such as changing an analog dead band value, turning on or off a breaker, or adjusting an actuator, as belonging to a role, as opposed to belonging to a person or some unnamed collection of people. The RTU model developed in this

dissertation has the following roles: Operator, Engineer, Vendor, Administrator, Display, and Enterprise. The roles are described in detail in chapter four.

Chapter two mentioned the Field Device Protection Profile for SCADA Systems in Medium Robust Environments published by NIST [48]. The main focus of this document is enumeration of Authentication, Authorization, and Audit capabilities for field devices, including RTUs. The NIST specification asserts that the RTU or field device must be able to enforce access control on protected data based on at least the following criteria: roles, location (of the subject), and time of day / day of week. Justification is given in some detail in the NIST document. The protection profile provides only guidance about what is expected and ignores implementation specifics completely; nor does it specify how access control constraints are to be specified.

To support these criteria and to further strengthen the RTU, the access control model includes additional constraints. The NIST PP identified the following constraints: location, time of day, day of week, and role. To that the RTU role based access control model adds the notion of state and the notion permission type. Constraints play an important role in the security of the RTU by facilitating secure operation through the application of the least privilege, allowing users to have only those permissions needed to carryout their operations. Roles, rather than the more traditional access control schemes, makes grouping permissions more logical and constraint on roles allows least privilege to be logical conceived.

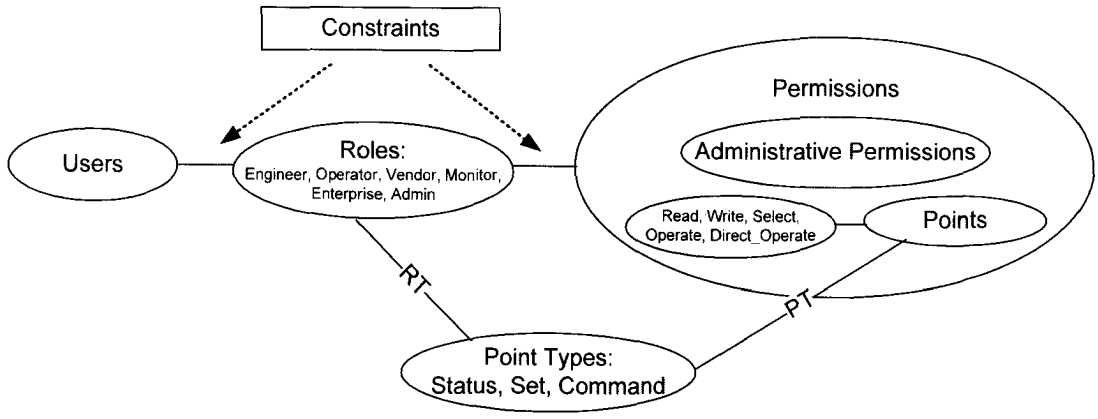


Figure 3.3. RTU access control model.

3.2.3 Reduced kernel OS

The second vulnerability that was identified in section 3.1 was commercial off the shelf (COTS) software, particularly COTS operating systems. A potential option is to eliminate this level all together by having a single process or program perform all of the duties of the RTU and essentially return to a runtime executive model. There are several reasons this is less than desirable. One is that the RTU application will have to handle on its own those functionalities usually provided by the operating system. It is also inefficient since a specific RTU application may have to be completely re-written to change what amounts to superficial changes, such as supported protocol. Most importantly it makes the task of software development much more difficult since those operating system abstractions are not available to the software developer. Also, as changes are made to the RTU program, the entire program must be re-verified, and it is not possible to re-use parts of an existing verifies solution without re-verifying them.

Recall from chapter two that the TCB is all of the hardware and software that is involved in enforcing a security policy. For commercial operating systems this is the entire operating system kernel, as well as the hardware (memory paging). The size of the code base, millions of line of code, and the incident of software flaws make it difficult to

trust the TCB. The key to a more trustworthy TCB is to reduce the size of the TCB. A smaller TCB is more trustworthy because

1. There is less code, and therefore less chance of flaws.
2. A smaller TCB is easier to understand.
3. A small TCB is potentially amendable to formal verification.

For RTUs, one way to achieve a smaller and potentially more trustworthy TCB, is to use a reduced kernel OS. Two approaches to creating a reduced kernel OS for RTUs are presented in chapter five. The first approach is to base the RTU on reduce COTS operating system. The second option is to use a microkernel or separation kernel to isolate multiple COTS operating systems and distribute the RTU components across them.

In the first option, the secure RTU model uses a reduced COTS kernel to provide only the minimal amount of functionality needed by the RTU for operation. By eliminating unneeded elements of the kernel, not only is its size reduced, but a large amount of code with potential flaws is also eliminated. A particular system like an RTU does not necessarily need all of the functionality of the typical desktop environment. By stripping the kernel of its unnecessary code, the threat that this code contains exploitable flaws is significantly lowered. The reduced kernel then serves as a more secure base on which SCADA and other RTU applications can be built.

A second option for the secure RTU model is to use a separation kernel [96;98] and the MILS architecture to isolate components of the RTU from each other. Individual components of the RTU (see figure 2.3) can be placed in separate partitions as shown in figure 3.4. There are several advantages of compartmentalizing the RTU in this way.

The compartmentalization prevents a software flaw in one system from modifying data, structures, or code in other modules. Also, if COTS components are used, they can be placed in their own partition and pose no security threat to other modules.

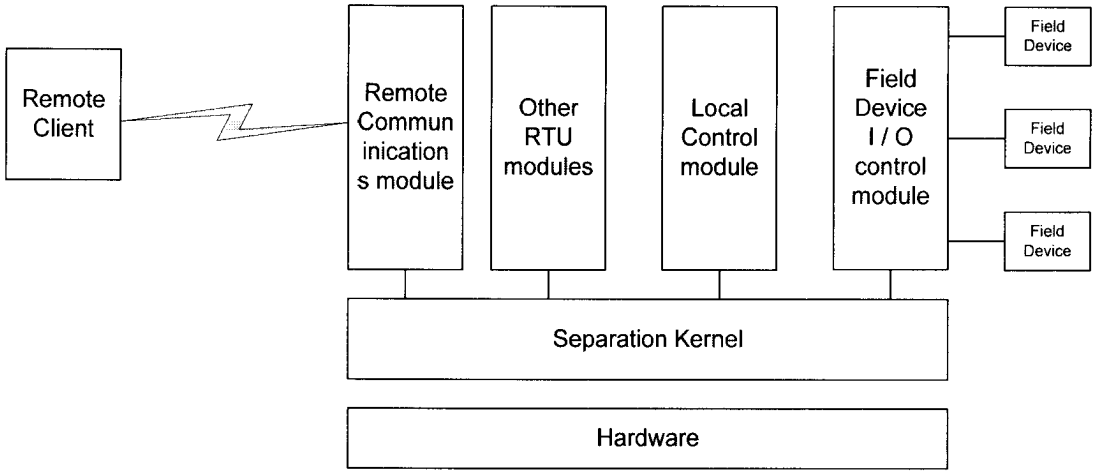


Figure 3.4. Secure RTU model with a separation kernel.

3.3 A security hardened RTU architecture

The RTU role based access control model must be enforced at some level in the RTU. The application layer is often the choice for RBAC systems, since they tend to deal with abstract permissions that are difficult to integrate into the operating system. Based on the reduced kernel approaches in section 3.2.3, especially the microkernel approach, enforcement of the RTU access control model at a more primitive level is possible and desirable.

Building on the microkernel concept of extracting and isolating system components and security functions in their own partitions, rather than including them in an application or in the operating system, a security-enhanced SCADA RTU architecture is derived. The model pictured in figure 3.5 has been developed to provide a high level description of the security enhanced RTU architecture. In this model only an input output (IO) controller has access to analog and digital IO ports. Access to status points

and command points is mitigated through the access control enforcement and security functions modules which provide a public interface for RTU services and share a private (trusted) communication interface for sharing security relevant information. All access to RTU points is through the access control enforcement layer, where an access control decision is influenced by both the access control policy and trusted security attributes obtained from protected and verified security functions. Security functions, particularly authentication are exported to remote users through the security enhanced DNP3 protocols described in section 3.1.

Besides authentication, an example of a secure function would be the access control decision function *check_access*, which takes as an argument a subject, an object and an operation, and returns true if the subject is allowed to carry out that operation, and false if not. The decision function *check_access* is called before attempting to access the protected RTU points. *Check_access* informs the access control enforcer that *check_access* has been called, by which subject, and the result of the function. The SCADA communication module can then access the point (provided *check_access* was successful). While it would be possible to have the enforcement module provide a single interface and make calls to the security mechanism internally the proposed method of separating the security functions should make them simpler, easier to verify, and interchangeable. In addition, it may be possible that a single call to the appropriate security functions could result in the enforcer granting multiple accesses to the caller.

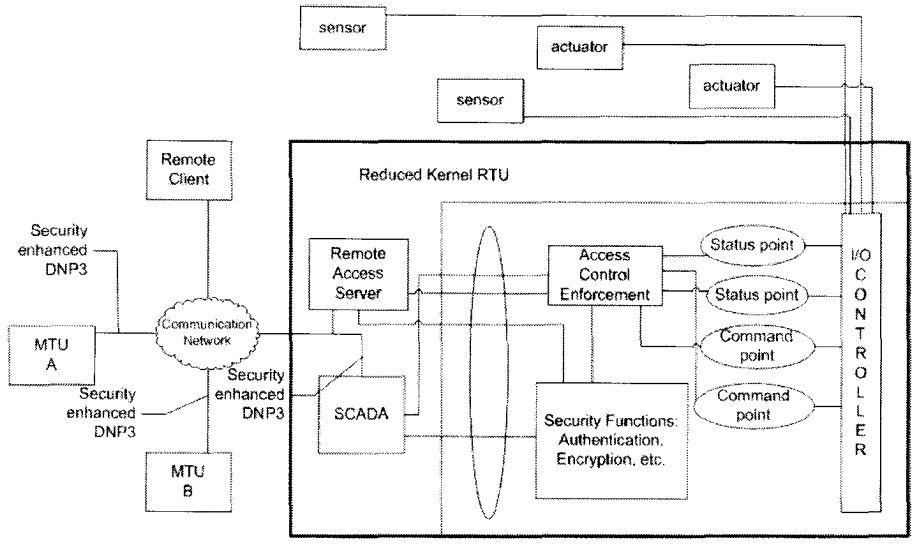


Figure 3.5. Security enhanced RTU architecture.

3.4 Conclusions

This chapter defined the security perimeter of the RTU and discussed the specific threat model for RTUs. In response to the threat model this chapter has introduced the concepts of the RTU access control model and reduced RTU kernels. These concepts along with a previously developed SCADA communication scheme are combined to create a security hardened RTU architecture. Roles and constraints will play a central role in defining the RTU access control model and micokernels and minimal COTS kernels have been identified as potential candidates for a reduced RTU kernel. In the next chapter, the RTU access control is explored in greater detail and the RTU access control model developed as part of this dissertation is defined. Chapter five explores in greater detail reduced kernels for RTUs.

CHAPTER IV

RTU ACCESS CONTROL MODEL

The lack of authentication in SCADA protocols leaves the RTUs and other field devices in a position of trusting that all received requests are from a valid and authentic source and should be dutifully carried out. As seen in chapter two a primary focus of SCADA security effort has been on authentication. But, as discussed in chapter three, providing authentication is just part of a comprehensive security architecture for SCADA systems, including RTUs. Providing robust and layered security requires that the RTU further protect itself by providing fine grained access control that assures authenticated users are only allowed to carryout authorized actions. This chapter presents the RTU access control model which has been developed through this dissertation research.

4.1 Core elements of the RTU access control model

The RTU role based model is based on the conventional role based access control concepts of users, who are assigned to roles and permissions that are assigned to roles. Specifically the model includes five core sets: users (U), roles (R), points (P), operations (O), and permissions (PERM). These sets have the usual meaning with some exceptions, specifically the replacement of the usual set of object (OBJ) with points (P) to reflect the RTU domain. These sets are fully defined in section 4.1.1. In addition there are two significant relations in the standard role based access control model, the user assignment

relation (UA) and the permission assignment relation (PA) which, described in section 4.1.2.

4.1.1 Sets

Subject (Users)

Subjects are the active entities of the system and of the associated requested operation. Subjects are traditionally understood as users, with the understanding that some agent within the computer system is acting on behalf of the actual user. For RTUs a subject may certainly be a human user, but we must also consider that the subject might be another computer system. For example, a HMI display might make requests to an RTU independent of any human interaction, or an MTU may collect information from an RTU automatically. In both these situations an external computer system is the subject or the user. Therefore, the notion of RTU subject includes both human users as well as remote systems (that are apart of the SCADA network) both of which must be acting through some local process or agent on the RTU.

Roles

Roles provide a logical means of grouping a set permissions, and convenient way to manage the assignment of permissions to users. Many RBAC models provide for an unlimited number of roles, which is appropriate for large organizations that can have many users playing many different roles. For RTUs the goal is different. The functionality of an RTU is very specific so a limited number of fixed roles will be acceptable. The RTU access control model provides the following roles:

- ***Engineer*** – The role of engineer captures the activities of SCADA engineers, who design and maintain SCADA system. They do not carry out day to day operations,

but analyze data and parameters off line, and update safety and monitor controls. Engineers need to have the ability to read most status points, including those most frequently used by operators as well as more obscure status points which engineers use to carryout safety and protection analysis.

- ***Operator*** – The operator role captures the activities of users who carryout day to day operation of the SCADA system. Operators need access to the status points that indicate the status of the current system, and any command points that they will use in response to observations about the entirety of the SCADA system.
- ***Display*** – The display role captures the activities of passive HMI machines. This role has only read access, and only to status data.
- ***Vendor*** – The vendor role captures external entities associated either the RTU or devices connected to the RTU. Vendors may need to observe certain values in order to provide technical support to system operators. However they should not be allowed to observe all points and should be prevented from changing command points or set points.
- ***Enterprise*** – the enterprise role address the current and suspected continuing trend that some systems or users from the enterprise network may desire access to RTUs. Users are internal entities to the organization that have an interest in some data values on the RTU. Or they might be business partners who are provided access to certain data as part of their arrangement with the owner of the SCADA system. This role will have limited permissions reflecting the fact that those network are less trusted.
- ***Admin*** – The admin role captures the administrative duties of the RTU. This user (or users) is responsible for defining roles and assigning users to roles. This role should

not have any access to RTU process related data points. This provides for separation of duty so that a user cannot both assign themselves roles and have access to the RTU system values.

Objects (Points)

The purpose of access control is to protect systems resources. For computer access control systems these system resources are generically referred to as objects. For general purpose systems such as PCs files are the most common abstraction to which access control is enforced. In role based access control, objects are generic and take on specific meaning only in the context of a target domain. The RTU roles based access control model is such a target domain. For RTUs and other field devices the primary abstraction is the data point. Recall from chapter two that RTUs are field devices that are typically connected to sensors and actuators that measure and control physical systems. Points are a universal abstraction in control systems, and refer to named or unnamed variables whose value: (1) directly relates to sensor readings or actuator settings, (2) is directly derived from one or more sensor reading values, or (3) directly influences the reading or writing of sensor and actuator values. Points are the digital representation of the telemetry and control provided by an RTU. Therefore, in the model, the standard RBAC object (OBJ) is replaced with points (P) where a point represents a single data value.

Operations

The operations with which we are most familiar are read, write and execute. These correspond to the basic operations for files on standard commercial operating systems, and for memory on most commercial hardware. Read and writing points is meaningful however, over the years SCADA protocols have developed their own

standard set of operations on points. These operations are *read*, *select*, and *operate*, and are described in more detail in [17]. These operations came about to compensate for possible errors in transmissions of early and less reliable serial communications. *Read* has its standard meaning; writing a value to a point is a two step operation usually referred to as select before operate. To write a value to an RTU point a user would first send a *select* operation, which identified the point to be written to and the value to be written. After receiving an acknowledgment, the user would then send the *operate* command which included the same point identifier and the value. Only if *select* and *operate* requests matched would the value be written to the actual point. The RTU access control model adopts the standard SCADA operations of *read*, *select*, and *operate* as its definition of *operations*.

Permissions

A permission is the approval to carry out some operation on one or more points. Permissions group the operations and points into sensible actions. For example select digital output one, or operate analog output two. Not all combinations of operations and objects are necessarily valid.

4.1.2 Relations.

Central to the role based access control model are the *user assignment relation* (UA) and the *permission assignment relation* (PA). The user assignment relation is a many to many relation that maps user to roles, such that a user can be assigned to more than one role and a role can be assigned to more than one user. The permission assignment relation is also a many to many relation between roles and permissions. In the core model these two relations establish the policy that is enforced by the model. For

every request r, where r = <subj, point, oper>, r is allowed if the user is assigned to at least one role ((user, role) is an element of UA) which has the permission (oper, point) (((oper, point), role) is an element of PA). This is formally defined as the function: *check_access*(u,p,oper).

4.1.3 Core model definition

The previous definitions are summarized in figure 4.1, which gives the core RTU role based access control model definitions.

Sets:

USERS	– The set of RTU users.
ROLES	– The set of roles defined for the RTU {Engineer, Operator, Display, Vendor, Administrator, Enterprise, Restricted}
POINTS	– The set of all points in the RTU
OPER	– a set of operations on P, {read, write, select, operate}
PERM	– The set of permissions of the form <oper,p>. PER $\subseteq$ OPER X P.

Relations:

UA – A relation of users to roles UA $\subseteq$ U X R.

PA – A relation of roles to permissions UA $\subseteq$ PERM X R

Functions:

roles(u:USERS) → ROLES a function that maps every user to a subset (possible null) of ROLES.

assigned_perm (r:R) → 2 ^ PERM mapping of roles onto permissions. *assigned_perm*(r) returns the permission assigned to role r.

Figure 4.1. Core RTU access control model definitions

4.2 Additional access control factors

In the core model that was just presented, the access control decision is based solely on the user assignment relation and the permission assignment relation. An advantage of this simplification versus the access control matrix model is the reduction in the number of relationships that need to be managed, from $O(mn)$ to $O(m+n)$. However the real advantage of role based access control is the potential to capture and

express constraints. As discussed in chapter two, constraints are one of RBAC's most important benefits. Constraints allow a model to capture and express accurately, complex and diverse policies; to better enforce separation of duty; and to achieve least privilege. In this section we consider possible criteria for constraints.

The NIST Field Device Protection Profile for SCADA Systems in Medium Robust Environments [48] is a draft document developed by PCSRF to provide a means for SCADA and industrial control system community to express the security requirements for the next generation of field devices and RTUs. The document asserts that the next generation of field device must implement access control, and furthermore that

> "*The access control decision shall be based on a variety of factors that are configured by an Administrator. The field device shall support at least the following access control factors: user role, system location, and time of day / day of week.*" [48].

The core elements of the model presented in section 4.1 provides only the role factor. To allow the additional factors listed in the NIST protection profile, location, time of day, and day of week can be achieved through the addition of constraints. In addition to the factors listed in the protection profile, this dissertation identifies two additional access control factors: point type, and system state. Sections 4.2.1 through 4.2.4 describe and define how each factor is to be interpreted in relation to the process control domain.

4.2.1 Location

For RTUs the location criteria applies to users (or systems) making requests of the RTU. The motivation behind the use of the location criteria is to allow the model to support limiting specific actions (or permissions) from originating from certain locations

irrespective of the role assigned to the user. For example, a breaker may only be turned on from the control room (but it might be turned off from anywhere), or a valve position may only be operated from the plant floor. The following is a list of locations, and reflects the locations currently supported in the model. The model can easily be extended to include other locations.

- Control Room
- Plant Floor
- Enterprise Campus
- Unknown

For the RTU to enforce restrictions based on location it must be able to associate a given user with a location. There are a number of different possibilities for associating a user with a given location. It is assumed in the model that structures needed to provide this association exist and are available. One possible choice is to bind an IP address to a particular subject at the instance of the request and then to use an established association of IP addresses to locations to determine the user's location. There are obvious short comings with this approach. An alternative would be to use GPS data and add location attribute information into a network communication protocol layer.

Definition 4.1. A set of potential user locations

LOCATION = {PLANT_FLOOR, CONTROL_ROOM,
ENTERPRISE_CAMPUS, UNKNOWN}

Definition 4.2. A user's location is given by the many to one relation

location $\subseteq$ USERS X LOCATION

The meaning of location(u , l) is that user *u* is currently in location *l*.

4.2.2 Time of day and day of week

Most RTUs operate 24 hours a day seven days week and 365 days a year. The primary reason for including time of day and day of week as potential criteria for access control decision is to limit access of users to correspond with their work schedules. For example, Alice works only on weekends and she is an engineer. The inclusion of time of day and day of week allows the security administrator to limit Alice's privileges to the weekend, when she is on duty; preventing her from accessing the system at an unexpected time. The security benefits of this are two fold, first this helps support least privilege and protects against a session hi-jack where an attacker continues to use Alice's credential after she has finished work.

Definition 4.3. The time of day is determined by the function *time_of_day* that returns a value indicating the current time of day defined by the set TIME_OF_DAY the set of minutes in twenty four hour day denoted as hh:mm. These are discrete times which can be enumerated. Rather than write each individual time, we also define a short hand ii:jj – kk:ll to denote the all discrete times between the interval ii:jj and kk:ll inclusive.

time_of_day() → td ∈ TIME_OF_DAY

TIME_OF_DAY = {00:00, 00:01, 00:02, . . . , 23:58, 23:59}

Definition 4.4. Day of the week is determined by a function day_of_week that returns a value indicating the current day of the week as an element from the set DAY_OF_WEEK

day_of_week() → dw∈DAY_OF_WEEK

DAY_OF_WEEK = {MONDAY, TUESDAY, WENSDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}

4.2.3 Point type

This dissertation identifies three possible types of points: *status points*, *control points* and *configuration points*. *Status points* represent the value read from a sensor, such as temperature, or possibly a derived value such as the deviation of a temperature reading from a static set value or the difference between two temperature sensors readings. *Control points* dictate, directly or indirectly, the behavior of connected actuators. A digital control point turns something on or off; an analog control point might dictate a valve position. *Configuration points* affect either status points or control points. For example a configuration point might dictate a dead band value for a sensor reading, or the frequency of a pulse width modulation control (which could be turned on or off by a digital control). Each point in the model is mapped to only one point type. In addition to the above point types there is a special permission type, *nil*, indicating that a point has yet to be assigned a type.

Definition 4.5. A set of point types

POINT_TYPE = {STATUS, CONFIGURE, CONTROL, NIL}

Definition 4.6. Each point is associated with only one point type defined by the many to one relation

point_type $\subseteq$ POINT X POINT_TYPE

The meaning of *point_type*(p, pt) is that point p is of point_type pt.

4.2.4 System state

RTUs and other field devices often have a set of states such that at any one time they are only in a specific state, and only certain operations should be carried out when in that state. For example, during start up operating some of the control points should be

prohibited. This may be safety related but clearly has security implications. In the model, the state of the system is global to the RTU, and changes to the state occur both as the result of internal processes (for example detection of the completion of the startup routine) and in limited cases as the result of human initiation, such as entering the maintenance state. The following states are used in the model:

- **Maintenance** – Indicating that the RTU is undergoing some type of maintenance activity
- **Operate Secure** – The RTU is operating, doing its normal activity, but is in an elevated security posture
- **Operating** – The normal operating state
- **Panic** – This state is entered when an error or failure is detected, errors and failures include security errors and failures (which ideally the RTU will be able to detect).
- **Recovering** – The RTU has experienced and error and is attempting to recover from the error.
- **Shut down** – This is the state entered when the RTU is told to shutdown or reboot.
- **Start up** – This is the state entered when the RTU is powered on.

Definition 4.7. The set of system states is defined by the following set

SYSTEM_STATE = {MAINTENANCE, OPERATING_SECURE,
OPERATING, PANIC, RECOVERING, SHUT_DOWN, START_UP}

Definition 4.8. At any given time the system is some state which is given by the function

system_state() → s ∈ SYSTEM_STATE

4.3 RTU constraints

The factors identified in section 4.2 can be incorporated into the RTU access control model using constraints. Recall from chapter two that there are several different possible types of constraints that the model might choose to express. Separation of duty constraints have been among the most popular, but chapter two identified seven different types or categories of constraints. The RTU access control model can support access control decisions based on the factors identified in section 4.2 by using exogenous constraints. Exogenous constraints are constraints whose attributes are not a part of the core RBAC model, and were defined in chapter two. The constraints will affect the relations in the model, specifically the UA and PA relations. Since these attributes will change during the runtime execution of the RTU, they are dynamic constraints. We incorporate into the model three types of constraints, role activation constraints, permission activation constraints, and point type constraints.

4.3.1 RTU role activation constraints

RTU role activation constraints place runtime or dynamic constraints on the use of one or more roles by a user. These are exogenous prohibition constraints, and define conditions when a user, who would otherwise have access, is prevented from making use of a role to which he or she has been assigned.

Definition 4.9. The RTU access control model limits the conditions under which a user may not use a given role to access some permission in the relation

RAC $\subseteq$ U X R X 2^{LOCATION X TIME_OF_DAY X DAY_OF_WEEK X SYSTEM_STATE}

The meaning of RAC(u, r, cs) is that a user u is prevented from using role r if location(u) $\in$ cs or time_of_day()$\in$ cs or day_of_week() $\in$ cs or system_state()$\in$ cs.

4.3.2 RTU Permission activation constraints

RTU permission activation constraints place runtime or dynamic constraints on the permissions that can be accessed through a given role. These are prohibition constraints in that they define conditions under which a permission cannot be accessed by a role. As an example, consider this natural language expression of an RTU permission activation constraint: "an engineer (role) may not change the dead band value of the pressure sensor stations_12_section_2 from the enterprise network or an unknown location." We now give a formal definition for permission activation constraints.

Definition 4.10. The RTU access control model limits the conditions under which a given role cannot access a permission normally assigned to it in the relation

PAC $\subseteq$ R X P X 2^{LOCATION X TIME_OF_DAY X DAY_OF_WEEK X SYSTEM_STATE}

The meaning of PAC(r, p, cs) is that any user u is not allowed to access permission p through role r if location(u) $\in$ cs or time_of_day()$\in$ cs or day_of_week() $\in$ cs or system_state()$\in$ cs.

4.3.3 RTU point type constraints

The final constraint that is added to the model is the point type constraint. These constraints are more akin to mandatory constraints, though not identical. The previous two constraints defined in section 4.3.1 and 4.3.2 can possibly be null, in which case the model is equivalent to the core model, as the constraints are non-existent. The RTU point type constraints are more static than the role activation constraints and the permission

activation constraints. The point type constraints limit the kinds of operations that can be assigned to roles. Recall from section 4.2 each point is assigned a type from the set POINT_TYPE. The point type constraint is a static obligation assignment constraint that requires each role be assigned a set of point types. Enforcement of this constraint prevents roles from being assigned any operation on points of a specific type. We now formally define the point type constraint as

Definition 4.11. The RTU access control model limits the assignment of permissions to roles in the relation

$$PTC \subseteq 2^{R} \times 2^{POINT\_TYPES}$$

The meaning of PTC(rs,pt) is that the roles in rs are assigned to the point types in pt. A role r may be assigned to an operation on point p if and only if $\langle r,pt \rangle \in PTC$, where pt is the point type of point p.

4.4 The RTU access control model

The previous sections have described and identified additional criteria on which access control decision should be based, and defined three types of constraints to add to the RTU access control model that reflect these additional access control factors. The final RTU access control model can be defined. It incorporates the constraints, and unites together the different access control factors. The model definitions are summarized in figure 4.2. The functional operations on the model are defined in table 4.1. For each function we define arguments, preconditions, and postconditions as they apply to the model.

Sets

USERS – The set of RTU users initially.

ROLES – The set of roles defined for the RTU {Engineer, Operator, Display, Vendor, Administrator, Enterprise, Restricted}

POINTS – The set of all points in the RTU

POINT_TYPES – A set of point types: {status, control, configuration}

OPER – a set of operations on P, {read, write, select, operate}

PERM – The set of permissions of the form <oper,p>. PER $\subseteq$ OPER X PERM.

APER – A set of administrative permissions, permissions that operate on the model {assign_role, add_user, assign_permission, assign_type_P(), delete_user, add_PAC, add_RAC, add_PTC}

LOCATION – A set of locations from which users generate RTU requests {PLANT_FLOOR, CONTROL_ROOM, ENTERPRISE_CAMPUS, UNKNOWN}.

TIME_OF_DAY – The set of all hour minute combinations for time of day, given as hhmm, (0001-2400)

DAY_OF_WEEK – The set of days of the week {Mo,Tu,We,Th,Fr,Sa,Su}

SYSTEM_STATE – The set of device states {Reboot, Start_up, Shut_down, operating, Operate_secure, Maintenance, Recovering, Panic}

Relations

RAC $\subseteq$ U X R X 2^{LOCATION, TIME_OF_DAY, DAY_OF_WEEK, SYSTEM_STATE}

PAC $\subseteq$ R X PERM X 2^{ LOCATION, TIME_OF_DAY, DAY_OF_WEEK, SYSTEM_STATE }

PTC $\subseteq$ 2^R X 2^POINT_TYPES <r,pt> indicating which point types a role may operate on.

UA $\subseteq$ U X R. A relation of users to roles

PA $\subseteq$ R X P. A relation of roles to permissions

PTA $\subseteq$ P X PT a many to one mapping of P to PT.

Functions:

roles(u:U) → R a function that maps every user to a subset (possible null) of R.

assigned_perm (r:R) → 2PERM mapping of roles onto permissions. *assigned_perm*(r) returns the permission assigned to role r

location(u) → l ∈ L a function mapping a user u to a location.

point_type (p) → pt ∈ POINT_TYPE is a function that maps a point p, to a point type.

time_of_day() → tod ∈ ToD a function that returns a ToD element representing the current time of day

day_of_week() → dow ∈ DoW a function that returns a DoW element representing the current day of the week

system_state() → ss ∈ D_S a function that returns the current state of the device.

role(rac:RAC) → r ∈ ROLES a function that returns the role of given role activation constraint.

user(rac:RAC) → u ∈ USERS a function that returns the role of given role activation constraint.

Figure 4.2. The RTU access control model.

Table 4.1. RTU access control operational functions

Function	Arguments	Preconditions	Postconditions
check_access	U, op, p, result	$u \in$ USERS $\wedge$ op $\in$ OPER $\wedge$ $p \in$ PERM	result = $\exists r \in$ roles(u) | (p,r) $\in$ PA $\wedge$ (u,r) $\in$ UA $\wedge$ ($\forall$ (cr,cp,cs) $\in$ PAC, $r \neq cr \vee p \neq cp \vee$ (location(u)$\notin$ cs $\wedge$ time_of day()$\notin$ cs $\wedge$ day_of_week()$\notin$ cs $\wedge$ system_state()$\notin$ cs $\wedge$) $\wedge$ ($\forall$ (cu,cr,cs) $\in$ RAC, $r \neq cr \vee u \neq cu \vee$ (location(u)$\notin$ cs $\wedge$ time_of day()$\notin$ cs $\wedge$ day_of_week()$\notin$ cs $\wedge$ system_state()$\notin$ cs $\wedge$)
add_user	User	user $\notin$ USERS	user $\in$ U
delete_user	User	user $\in$ USERS	user $\notin$ U $\neg\exists$ r:R|(user,r) $\in$ UA
assign_user	user, role	user $\in$ U $\wedge$ role $\in$ ROLES	(user,role) $\in$ UA;
deassign_user	user, role	user $\in$ U $\wedge$ role $\in$ ROLES $\wedge$ (user,role) $\in$ UA	(user,r) $\notin$ UA;
assign_role	role, op, obj	role $\in$ ROLES $\wedge$ (op,obj)$\in$ PERM $\wedge$ (role, point_type(obj)) $\in$ PTC	((op,obj),role) $\in$ PA
deassign_role	role, op, obj	role $\in$ ROLES $\wedge$ (op,obj)$\in$ PERM $\wedge$ ((op,obj),role) $\in$ PA	((op,obj),role) $\notin$ PA
assign_type	p, pt	p $\in$ POINTS $\wedge$ pt $\in$ PT $\wedge$ point_type(p) = nil	Point_type(p) = pt
add_RAC	roles, perms, cs	roles $\subseteq$ R $\wedge$ perms $\subseteq$ PERM $\wedge$ cs $\subseteq 2^{\{\text{LOCATION} \cup \text{TIME\_OF\_DAY} \cup \text{DAY\_OF\_WEEK} \cup \text{SYSTEM\_STATE}\}}$	cs $\in$ RAC
add_PAC	users, perms, cs	users $\subseteq$ U $\wedge$ perms $\subseteq$ PERM $\wedge$ cs $\subseteq 2^{\{\text{LOCATION} \cup \text{TIME\_OF\_DAY} \cup \text{DAY\_OF\_WEEK} \cup \text{SYSTEM\_STATE}\}}$	cs $\in$ PAC
add_PTC	roles, types	roles $\subseteq$ ROLES $\wedge$ types $\subseteq$ POINT_TYPES $\wedge$ (roles, types) $\not\subset$ PTC	(Roles, types) $\in$ PTC
remove_RAC	roles, perms, cs	roles $\subseteq$ ROLES $\wedge$ perms $\subseteq$ PERM $\wedge$ cs $\subseteq 2^{\{\text{LOCATION} \cup \text{TIME\_OF\_DAY} \cup \text{DAY\_OF\_WEEK} \cup \text{SYSTEM\_STATE}\}}$ $\wedge$ cs $\in$ RAC	cs $\notin$ RAC
remove_PAC	users, perms, cs	users $\subseteq$ USERS $\wedge$ perms $\subseteq$ PERM $\wedge$ cs $\subseteq 2^{\{\text{LOCATION} \cup \text{TIME\_OF\_DAY} \cup \text{DAY\_OF\_WEEK} \cup \text{SYSTEM\_STATE}\}}$ $\wedge$ cs $\in$ PAC	cs $\notin$ PAC
remove_PTC	roles, types	roles $\subseteq$ ROLES $\wedge$ types $\subseteq$ POINT_TYPES $\wedge$ (roles, types) $\in$ PTC	(roles, types) $\not\subset$ PTC

4.5 Check Permission Algorithm

A central model component is the function check_access(), Check_access determines whether a given subject can carry out a given operation on a given object. Table 4.1 gives the post conditions for check access. Check_access is modeled as a function and an algorithm for implementing check access is shown in figure 4.3

```
Function Check_access (subject, operation, point)
Begin
      allow_flag = false;
      PERM = to_perm(operation, point) // return the permission
                                       // associated with the operation
      UR = roles(subject)
      PR = roles(permission)
      UCR = {}
      PCR = {}
      CS = {location(user), TOD(), DoW(), State()}
      For each rac in RAC
            If user(rac) == subject
            For each cs in CS
                  If  cs ⊆ constraints(rac)
                  UCR = UCR ∪ role(rac);
      For each pac in PAC
            If PERM == perm(pac)
            For each cs in CS
                  If  cs ⊆ constraints(pac)
                  PCR = PCR  ∪ role(pac)
      UR = UR ∩ UCR
      PR = PR  ∩ PCR
      R =  UR ∩ PR
      IF |R| > 0
            Allow_flag = true
      Return allow_flag
end
```

Figure 4.3. Check_access algorithm.

The check_access function first identifies permissions associated with the requested operation, and then finds the roles assigned to the user, and the roles to which the specific permission has been assigned. Next the context functions `locations(user)`, `time_of_day()`, `Day_of_week()`, and `system_state()` are evaluated as stored as elements of the set CS. Then the

check_access function identified each role activation constraint from RAC that applies to the subject. If a role activation applies to a user, then a test is performed for each element of the constraint set CS. If an element is a subset of the current RAC's constraint set, then the RAC's role is added to the set UCR. The same process is repeated for the PAC using permissions and the set PCR. Finally any role in UR that is also in UCR is removed from UR and any role in PR that is also in PCR is removed from PR. If the intersection of UR and PR has at least one role then the subject is allowed to carry out the operation on that object.

4.6 Conclusions

This chapter has presented a formal model for RTU access control. The model is based on the core RBAC defined by NIST but without sessions. In the model, the primary factor for access control is roles, but additional context factors are allowed to influence the access control decision. These factors are expressed as constraints, and are given in the sets RAC, PAC, and PTC. The access control decision is provided by the function check_access and an algorithm for the check_access function is given.

CHAPTER V

RTU PROTECTION AND REDUCED KERNELS

The RTU access control model described in chapter four provides a model for determining authorized RTU actions. Application of the model to an actual RTU requires an authorization system and protection architecture. A typical authorization architecture includes a policy enforcement point (PEP), and a policy decision point (PDP) as shown in figure 5.1. As discussed in chapter two, the operating system plays a central role in security since it provides the interface to system resources and controls all access to system resources. Unfortunately, commercial operating systems tend to provide discretionary access control that is very coarse. Commercial operating systems typically follow the Unix model, where all system resources are treated as files. Processes (subjects) and files are assigned to a user id and a set of permission bits attached to each file determines whether a subject can access the file (resource). Subjects, not the system, determine who has access to the resources they control. This protection architecture is insufficient to enforce the RTU access control model from chapter four for two reasons: first, it is too coarse, because the model requires the ability to restrict access to specific operations on individual analog and digital IO points; second, the controls are discretionary allowing them to be arbitrarily changed. An alternate RTU protection architecture is needed to allow policies described by the model to be enforced.

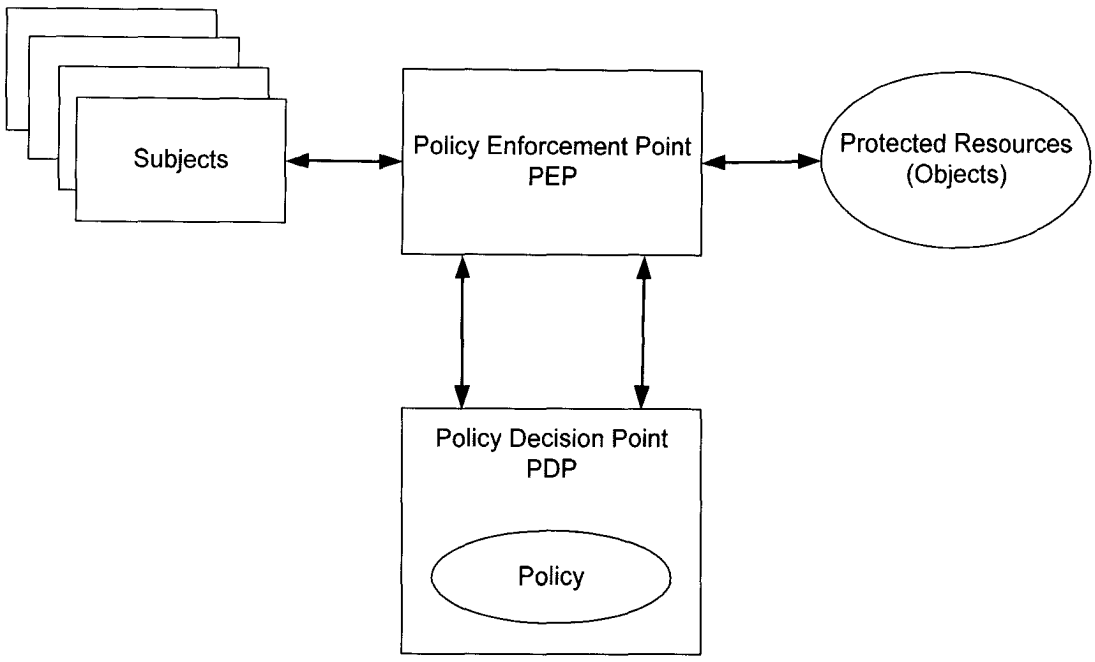


Figure 5.1. Typical authorization architecture.

Since the operating system level of abstraction is not sufficiently fine-grained, a logical choice is to place the PEP in a middleware layer and force all access requests through the PEP. This middleware layer will provide access to the RTU operations such as reading and writing digital and analog IO points, and other operations such as cold and warm restarts. It is possible to place the PEP within a specific application such as the DNP3 application layer process. The problem with locating the PEP and PDP within a single application is that in the case that the RTU supports multiple SCADA protocols, the RTU access control model will have to be implemented in each application. This is wasteful and inefficient, and most importantly can lead to an inconsistent application of a given policy. A middleware PEP placement is far superior as it allows centralized administration and enforcement, and assures that all software components are subject to the access controls.

The problem that faces a middleware layer PEP brings us to the second RTU threat area identified in chapter three – COTS operating system vulnerabilities. The middleware layer relies on the protection mechanisms in the operating system to ensure that all subjects must access the protected objects through the PEP. An operating system vulnerability could allow the PEP to be bypassed, and thus for the security of the RTU to be circumvented. The ability to circumvent the access controls is not limited to SCADA applications, such as the DNP3 server. Other RTU applications, even those not intended to access analog and digital IO points at all, could be allowed to read or make changes if they are able to exploit a vulnerability in the kernel. Clearly this is an undesirable situation, and given the poor track record with respect to security that is characteristic of commercial operating systems, it must be considered a strong possibility.

The two contributing factors to the poor security of commercial operating systems identified in chapter two included the size of the code base and their monolithic design. This chapter presents two approaches to creating a reduced kernel based OS for RTUs that strengthen the protection provided by an RTU middleware PEP. The first approach acknowledges that there is strong motivation to use commercial operating systems due to the significant cost savings that can be realized by both the cost savings of the OS itself and the savings in the time spent on application development. The approach leverages the fact that a good deal of the code base in commercial operating systems is not needed by the RTU, by proposing to reduce a COTS kernel to a minimal COTS kernel for RTUs. The second approach takes a more radical approach and proposes using a microkernel to isolate RTU components and place an isolated PEP between RTU resources and user level applications. The two minimal kernel approaches identified in this chapter reduce

or potentially eliminate the COTS OS threat faced by RTUs, and provide assurance that a middleware layer PEP for the RTU access control model is not subverted.

5.1 Minimal COTS kernel based RTU

Recall from chapter two that one of the motivating factors in the adoption of COTS components in SCADA environments was the need to contain costs. The majority of SCADA networks are privately owned and operated and in most cases are for profit companies, such as PG&E, American Water Works, W. R. Grace, and Proctor and Gamble. Vendors of SCADA components such as RTUs are strongly motivated to keep the cost of their products down and to keep the cost of ownership down as well. For this reason COTS operating systems, particularly Linux, are an attractive choice for SCADA devices, since their cost can be significantly less than the cost of developing a custom OS for the device (Linux further reduces cost by eliminating licensing cost associated with devices). However, as was discussed in chapter two, commercial operating systems' track record with respect to security is less than exemplary.

To address this shortcoming while maintaining the economic benefits of using a COTS OS, the minimal RTU kernel is achieved through a radical reduction of a standard COTS kernel. The goal of the radical reduction is to provide only the minimal amount of functionality needed by the RTU for operation. By eliminating unneeded elements of the kernel not only is its size reduced, but a large amount of code with potential flaws is also eliminated. A particular system like an RTU does not necessarily need all of the functionality of the typical desktop environment. By stripping the kernel of its unnecessary code, the threat that this code contains exploitable flaws is eliminated. The

reduced kernel then serves as a more secure base on which SCADA and other RTU applications can be built.

One of the primary sources of vulnerabilities in today's commercial kernels are device drivers. These pieces of code have a higher flaw rate, and in monolithic kernel design, complete access to the text and data sections of the kernel. To make commercial operating systems able to function on a large number of environments modern kernels include many drivers, filling the kernel with unneeded and possibly exploitable code. Unneeded device drivers are an excellent target for removal in creating a minimal COTS kernel RTU, as unneeded drivers can be permanently eliminated. It should be possible to greatly reduce the kernel using this approach as device drivers often represent about 70% of the operating system [84].

Besides drivers, there are also a number of support structures in a standard COTS kernel, the main ones being a file system, process model, interrupt handler, memory allocation, and scheduler. While these main components will need to remain in the kernel to function properly, it is possible to pare them down by removing unneeded lower level components. For example, support for many different file systems, such as NTFS are likely not needed by RTUs since they are characterized as stand alone systems.

The objective of the minimal commercial kernel RTU is to allow RTU vendors to capitalize on the cost savings that are provided by commercially available operating systems, while at the same time minimizing the security vulnerabilities. Any COTS OS is likely to have both known and unknown vulnerabilities. Those vulnerabilities are the result of software flaws in the OS code. Reducing the kernel reduces the lines of codes in the kernel and thus possibly eliminates the code containing unknown flaws. A recent

study [110] showed that code contains on average between 6 and 16 bugs per 1000 lines of code (loc). Therefore eliminate enough lines of code and you significantly reduce the occurrence of bugs. A RTU security architecture can then be constructed using the reduced RTU kernel and a middleware PEP as shown in figure 5.2.

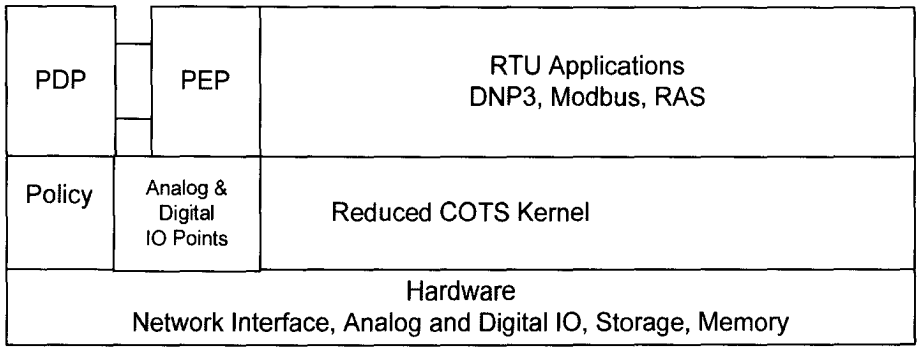


Figure 5.2. RTU security architecture using radically reduced RTU kernel.

5.2 Microkernel based RTU

The minimal COTS RTU kernel can eliminate many potential vulnerabilities. It is however still based on the monolithic kernel design, which means that all of the services provided to the RTU still exist in a single address space with little protection from each other. Particularly the analog and digital IO modules are not protected from other kernel objects or threads. Malicious code can still potentially break into the kernel by exploiting a vulnerability and gaining root access. Once root access is achieved an attacker can bypass the PEP and potentially directly manipulate analog and digital IO.

Truly robust RTU protection cannot rely only on the discretionary access controls enforced by commercial OSs. Instead a different approach is needed. The separation kernel and the MILS architecture, described in chapter two provide inspiration for a solution. Using this architecture, critical RTU resources, specifically analog and digital IO ports and modules can be truly isolated from other RTU components. Individual

components of the RTU (see figure 2.3) can be placed in separate partitions as shown in figure 5.3.

The security advantages of strongly isolated RTU components are numerous. First, the compartmentalization prevents a software flaw in one system from modifying data, structures, or code in other modules. This type of isolation helps the RTU achieve availability; this is essentially the MILS fault isolation goal. Another advantage is that, if COTS components are used, they can be placed in their own partition and pose no security threat to other modules. For example, if an RTU vendor chooses to include a web interface, the server software can be placed in its own isolated partition, and the system architecture prevents the possibility of the web server being compromised and used to subvert the RTU.

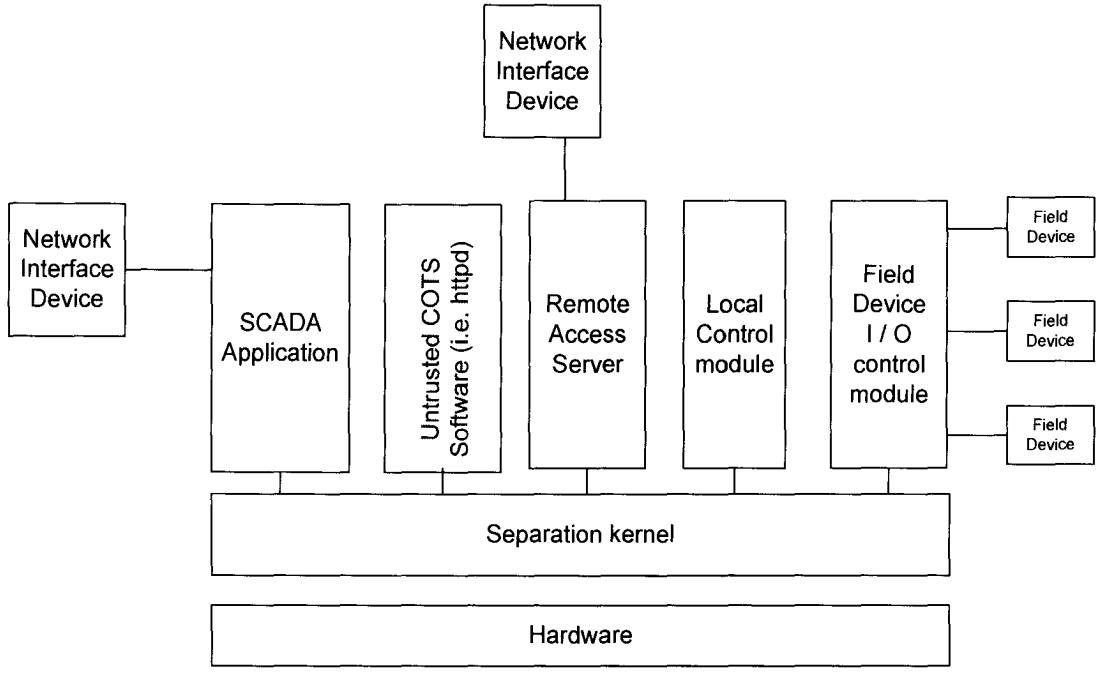


Figure 5.3. MILS RTU with isolation of RTU components.

Complete isolation of each component is not possible, there must be some cooperation among the isolated component for the RTU to function. For example, the

communication module, which sends and receives SCADA messages, will need to retrieve actual IO data from the analog and digital IO modules, and possibly pass values back to these modules. The complement to isolation is cooperation or sharing. The security achieved by isolating RTU components can easily be undone when the various components are allowed to interact. Consider the analog and digital IO components. These are drivers that access the RTU's IO ports. Isolation allows the ports and memory to be dedicated to an analog input module, or digital output module. The dedication of IO ports to IO modules prevents other potentially misbehaving modules from accessing IO ports, or modifying the programs of IO modules. However, for the RTU to carry out its function, some other isolated components will need to have access to the input and output values from the IO modules.

Recall that the separation kernel is a microkernel and that a microkernel provides three primary abstractions: address space, execution, and inter process communication (IPC). The isolation presented so far is represented by address spaces. The execution abstraction is achieved through tasks, which represent a unit of execution. Finally IPC allows tasks, which are isolated in one or more address spaces, the ability to cooperate and share information. Tasks communicate with each other through a set of IPC primitives provided by the kernel. Moreover the kernel, as the arbiter of task IPC, can determine if two tasks are allowed to send and receive messages. In MILS there is an additional layer called the partition communication system, which extends the IPC functionality with security primitives.

While some systems may choose to prevent any partition from communicating with any other partition, it is through secure cooperation that robust, secure, and useful

systems can be built. The advantage of MILS and the structure of the IPC or PCS is that the system architect can designate which partitions or tasks are allowed to communicate with each other, and the separation kernel enforces this absolutely. This allows application layer reference monitors to be placed between different partitions, and assure that these reference monitors are not bypassed. Thus, rather than relying on the kernel to make all access control decisions, the MILS architecture allows access control to be layered. User land applications that provide security are inserted between isolated application components, and the separation kernel guarantees that these components cannot be bypassed. Applying this principle to the security hardened RTU allows for the creation of one or more PEP to protect different isolated RTU components. Furthermore the PDP can be isolated as well, making it impossible for other RTU components to modify the policy. Figure 5.4 shows how the RTU PEP is inserted between various isolated RTU components. This allows the RTU to unequivocally apply its security policy to all executing entities on the system.

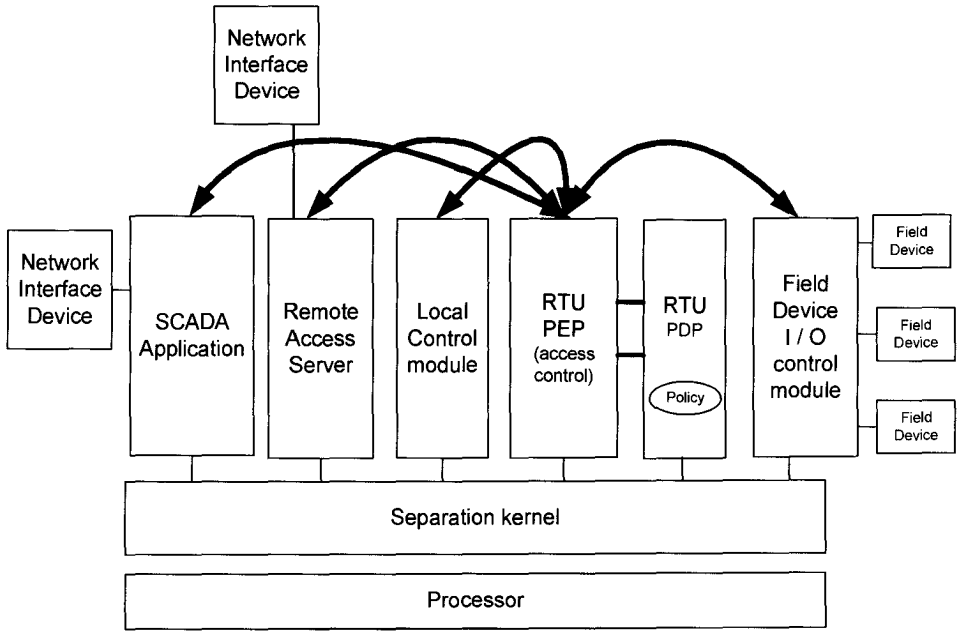


Figure 5.4. MILS RTU with PEP.

There are some disadvantages to applying the MILS architecture and a separation kernel to RTUs. The MILS architecture is targeted for high-assurance systems, in particular those that need to support multi-level security (MLS). While MLS is needed for many military systems, for SCADA systems, specifically for RTUs, it is not representative of the desired security objectives. MLS is a based on the Bell LaPadula model, which is a confidentiality security model. As mentioned in chapter two, for SCADA systems, availability and integrity are of primary importance, not confidentiality. Two other significant disadvantages to the separation kernels and the MILS architecture are cost and availability. A true separation kernel, which was first proposed in 1982, has yet to be developed. Recent advances in microprocessors are making the performance cost associated with a separation kernel more acceptable. As discussed in detail in chapter two, research and development of the MILS architecture and a separation kernel are currently active. Lynxworks and Greehhills software are both working on separation kernels that can achieve EAL seven certification, which is a key milestone to realizing the MILS architecture. However, to this author's knowledge no EAL seven certified separation kernel is yet available. Moreover, if a separation kernel is to become available, it will surely have a high cost associated with it, at least initially. Since SCADA operators face economic pressure to contain costs, the expense of a full separation kernel is likely to be out of their reach for the foreseeable future.

5.2.1 Alternative microkernel for the security hardened RTU

While clearly a certified EAL seven separation kernel provides the best microkernel for an RTU from a security perspective, microkenels in general can enable the development of more robust security, and provide partitioning and encapsulation that

are core to the MILS approach. A key difference being that they lack formal verification and the MILS architecture in which to be integrated. There are a number of microkernels available, several of which are open source and could be used in developing a microkernel based RTU. These are discussed in chapter seven where development of a microkernel based RTU prototype platform is presented.

5.2.2 RTU protection architecture using a microkernel

Given the obstacles to a MILS / separation kernel based RTU and availability of open source microkernels, a viable and promising alternative to a MILS based RTU is a microkernel based RTU. Although this alternative cannot provide the same high assurance as MILS, it can provide a superior RTU security architecture, and one that can potentially eliminate or mitigate the vulnerabilities in COTS operating systems and software in the RTU. It is this approach that is advocated here. The availability of open source microkernels is ideal for the development of secure RTUs. The three microkernels mentioned in the previous section, Fiasco, Pistachio, OKL4, are all sufficiently mature to support investigation of a microkernel based security hardened RTU.

The microkernel based RTU security architecture isolates RTU components by assigning each to its own address space. Each individual component then provides one, or more, interfaces for receiving IPC messages. The contents of the messages are dependent on the particular component. For example, the analog input component might accept a read message indicating the analog input point value that is desired. The component would then put the value in a response message. Another example might be the communication interface accepting an open connection message, or a send or a

receive message. The model includes a number of RTU components, such as local control, which might not necessarily be present in every RTU. It should be easy to accommodate this, as these can just become null address spaces, and give null responses to errant requests.

The microkernel provides for isolation of these components, providing the first layer of security. Next, security components, especially a PEP and PDP that capture the application of the RTU access control model described in chapter four, are woven into the components. The main security component is the RTU security service component. This component maintains the security related information, policy, and state information relevant to RTU security. Isolated components can send messages to the RTU security service. Rather than a single PEP, the microkernel based RTU has multiple PEPs in the form of guards that intercept communication between different components, and enforce the RTU security policy on these requests and replies. The security architecture is shown in figure 5.5.

5.3 Conclusions

This chapter has discussed protection architectures for the RTU that support the RTU access control model described in chapter four. The access control model is fine-grained and cannot be natively supported in most commercial kernels. Therefore some middleware protection scheme is needed. This highlights the threat posed to RTUs by COTS operating systems, since flaws in the OS can allow middleware protection mechanisms to be circumvented. Two approaches to reducing this threat were described in this chapter – a minimal COTS RTU kernel, and a microkernel. The minimal COTS kernel is proposed as a way to reduce the threat of vulnerabilities, but still allow SCADA

vendors and SCADA operators to capitalize on economic benefits that make COTS desirable. The MILS architecture and separation kernel serve as the inspiration for the microkernel approach. But rather than actually advocate the use of the MILS architecture and separation kernel, which are not readily available and are expected to be cost prohibitive for SCADA operators, the use of an open source microkernel is advocated, as it provides the benefits of MILS and separation kernel, though lacking formal verification, without the costs.

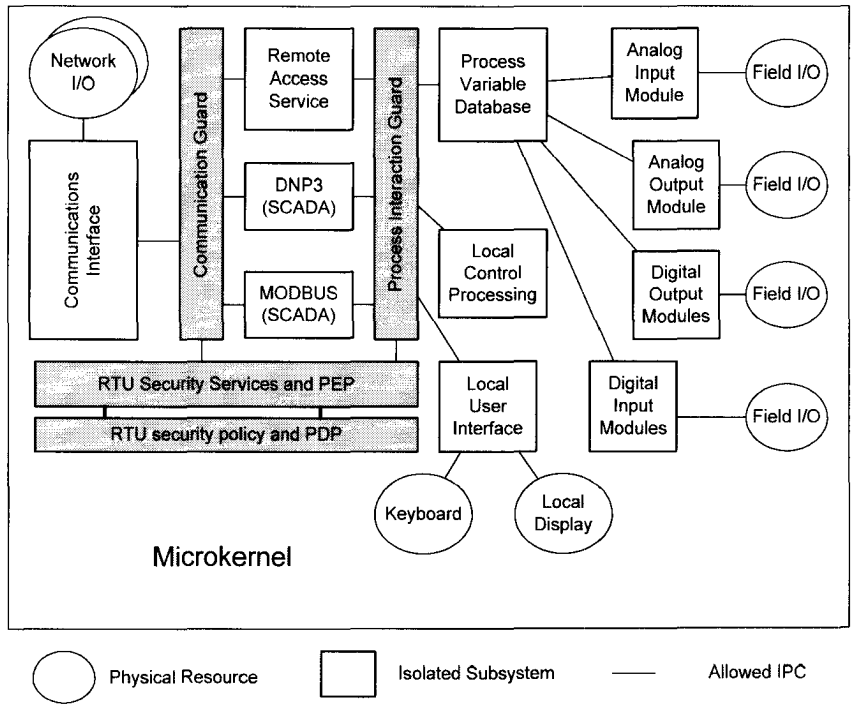


Figure 5.5. Microkernel based RTU security architecture.

CHAPTER VI

PROTOTYPE DEVELOPMENT AND TESTING

A new access control model for RTUs was presented in chapter four, and chapter five described two reduced kernel approaches for strengthening RTU security. This chapter describes the development and testing of a prototype hardened RTU. The prototype implements the developed RTU role based access control model as a middleware layer available to other RTU processes, and uses a reduced COTS kernel as described in chapter five. A security enhanced DNP3 protocol similar to that described by Patel in [29] was included to provide SCADA access to the prototype. RTUs and other industrial controllers usually have less available memory and processing power than traditional computing systems and have different performance requirements as well. The prototype was developed on actual RTU hardware from Sixnet and evaluated in a test bed environment including actual SCADA hardware. Both performance analysis and security testing were conducted in the prototype evaluation.

6.1 Prototype development platform

An mIPM form Sixnet was used as the prototype development platform. SixNet IO is a US based company that manufactures and distributes hardware for the industrial automation industries including: oil and gas production and distribution, water and wastewater treatment, and transportation. Their product list includes DCS controllers, IO modules, Dataloggers, and RTUs that support Ethernet, RS232 and RS485

communication standards. Their products are designed for harsh industrial environments and are Deutsches Institut für Normung (DIN) rail mountable. Sixnet also caters to OEMs, providing them with private labeling of controllers and RTUs. Sixnet's Linux based IPm controllers are open systems, allowing OEMs to create customized RTUs using open source tools, making them an excellent choice for prototype development.

6.1.1 Hardware

Sixnet RTUs and DCS controllers are actually embedded computers based on a PowerPC CPU with a 32 bit data bus operating at 50 Mhz. Besides the processor, RTUs and DCS controllers also have flash memory, dynamic RAM, and persistent (battery backed) RAM and come with a mix of 10/100 Ethernet ports, RS232 and RS485 ports as well as on board digital and analog IO. Battery backed RAM is common in industrial devices and is used for logging purposes since the contents persist through power cycles. Access to dynamic RAM on Sixnet devices is made available through an emulated disk.

6.1.2 Software

Sixnet offers RTUs and DCS controllers with either Winows CE or Linux operating system (OS). The IPm series of RTUs by Sixnet uses the Linux 2.4 kernel, and the Sixnet Linux installation includes a TCP/IP stack, common Internet daemons including telnet, echo, ftp, and daytime. Since the Linux OS is open source, the kernel source used by Sixnet is available. Basic OS utilities such as `cat`, `more`, `ls`, and `ps` are provided by `busybox`. The standard installation also includes `gdbserver` to allow remote debugging of applications.

6.1.3 Development Environment

To encourage OEMs to use their product, Sixnet provides a development package called IADK (IPm Application Development Kit). The kit includes a gcc cross compiler toolchain for compiling code that can be executed on an IPm based RTU. The kit also includes a client side GUI based debugger (insight) that connects to a running instance of the gdbserver on an IPm. In addition the kit includes static and dynamic versions of commonly used libraries. Finally the kit includes Sixnet's proprietary library used to access local digital and analog IO on the IPm. Using the library a program can read or write to local analog and digital IO by calling one of two functions: `IODBRead( . . . )` and `IODBWrite( . . . )`.

6.2 Hardened RTU prototype development

The most basic open IPm based RTUs from Sixnet was chosen as the platform on which to develop the hardened RTU prototype. Since the mIPM is a commercially available RTU, computing resource available to the prototype RTU such as dynamic memory, flash memory, and processor speed, were considered comparable to current commercial units. Sixnet's support for OEMs and OEM customization of mIPM products makes development and testing possible and provides a potential path towards future commercial deployment. The chosen RTU, designated by Sixnet as mIPM VT-241D was the least expensive RTU which had a nice complement of on board IO. The mIPM RTU has 16MB of flash, 16MB of dynamic memory, one auto detection 10/100 Ethernet port, and four serial ports. The on board IO consists of twelve binary inputs, four binary outputs, six analog inputs and two analog outputs. The binary inputs are

based on the mIPM's power supply which for the prototype was fifteen volts. The binary outputs can draw on the power supply as well, up to two amps. The technical specifications sheet for the mIPM are included in appendix D.

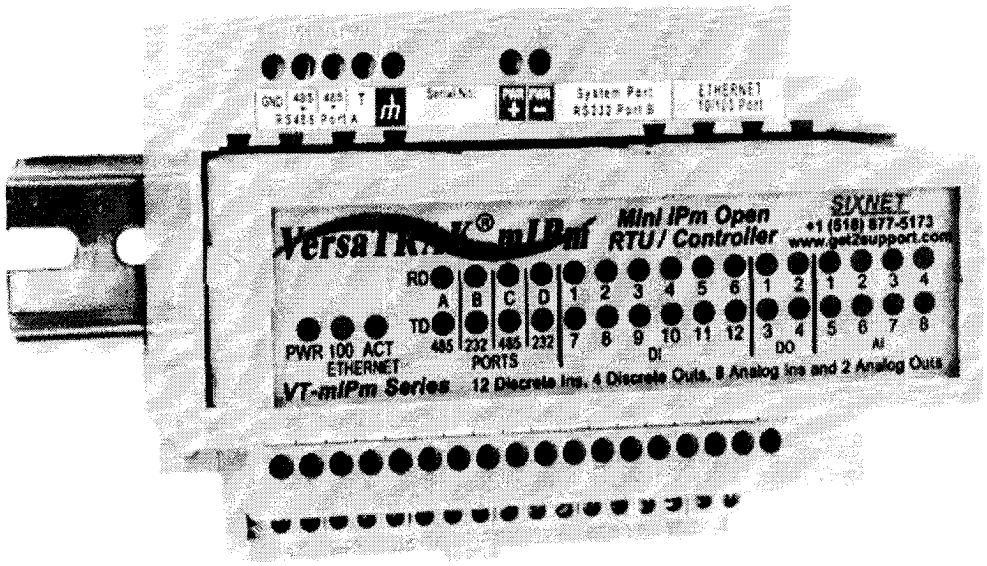


Figure 6.1. SixNet RTU used for prototype development source [111].

The prototype consists of three primary software components: first, a reduced COTS kernel, as described in chapter five, second, an implementation of the new RTU role based access control model described in chapter four which provides access to RTU IO points to all other RTU software, and third, an implementation of DNP3 enhanced with security measures, particularly authentication, to allow other SCADA devices to access the RTU.

6.2.1 Prototype development: Reduced Linux Kernel

The first component of the hardened RTU to be developed was the reduced kernel. Two approaches were identified in chapter five, a reduced COTS kernel and a microkernel. The reduced COTS kernel approach was chosen for this prototype. The use of a microkernel is explored in chapter seven. Some initial experimentation with reduced

kernels was carried out using the LynxOS [112]. LynxOS is a real-time operating system based on Linux and designed for embedded systems development. The environment supports the POSIX® standard and provides cross-compilation platform and kernel customization. A reduced kernel image was created using LynxOS's kernel customization features and the creation of KDI's or kernel downloadable images. The main feature of the kernel customization is the inclusion and exclusion of different drivers and supporting libraries. Other elements of kernel reduction were focused on setting variable values that affect performance such as the maximum number of processes, the size of the disk cache, or the maximum number of mounted file systems. To further reduce the size of the kernel, the libraries for IPv6 and the NFS were also excluded.

LynksOS support for the PowerPC was limited, and was not available for the mIPM. The Linux 2.4 Kernel is supported for the mIPM hardware, with a patch available from Sixnet. The source code for the 2.4 kernel was downloaded from kernel.org. The Sixnet mIPM is built around a Power PC based single board computer (SBC). In order for the kernel to run on the mIPM a patch to the kernel source must be applied. The patch is freely available from Sixnet.

After patching the kernel a reduced kernel image was created. This was achieved by eliminating a host of drivers and supporting software. Support for the following file systems was eliminated: REISER, HFS, BFS, EXT3, FAT, MSDOS, UMSDOS, VFAT, NTFS, EFS, NFS, and MINIX. Parts of the network libraries were eliminated including support for IPv6 and netfilter support in the kernel's TCP/IP stack. The final kernel configuration file is listed in appendix A. The final size of the reduced kernel was 1.3

megabytes. For comparison purposes a default installation of Redhat 7.3 which uses the 2.4 kernel was evaluated. The default Redhat 7.3 Linux kernel was 2.9 megabytes. The hardened RTU's reduced Linux kernel is less than half the size of a comparable standard Linux kernel. Assuming that software vulnerabilities are evenly distributed, this reduced kernel RTU has reduced by about one half, the number of kernel vulnerabilities that result from kernel code flaws, and which could present RTU attack vectors. Table 6.1 shows the sizes of the compressed and uncompressed kernel images and breaks down their size into text, data, and bss segments. Once the custom reduced kernel was compiled it was loaded onto the mIPM RTU.

Table 6.1. Reduced kernel and standard kernel comparision.

	Kernel Size				
	Uncompressed	compressed	text segment	Data	Bss
Hardened RTU Prototype Reduced Kernel	1.3 M bytes 1350116 bytes	553943 byes 541 K bytes	1285640 bytes	59592	13827
Standard Linux kernel (Red Hat 7.4)	2.9 Mbytes 3072843	1.2 Mbytes 1262048 bytes	1969848 bytes	431652	383452

6.3.2 Prototype development: RTU Role Based Access Control

After the new reduced kernel was loaded, the next hardened RTU component needed was an implementation of the RTU role based access control model described in chapter four. The RTU role based access control component was created for the mIPM RTU using Sixnet's IADK. The IADK library routines provide access to the prototype's hardware circuitry for analog and digital input and output. The access control scheme was implemented as a middleware security layer between the IADK library calls and a

new inter process communication (IPC) defined interface. Figure 6.2 shows the basic architecture of the implementation. The defined interface is available to any RTU user process or application through IPC calls described below. A permission was created for each primitive in the IDAK library.

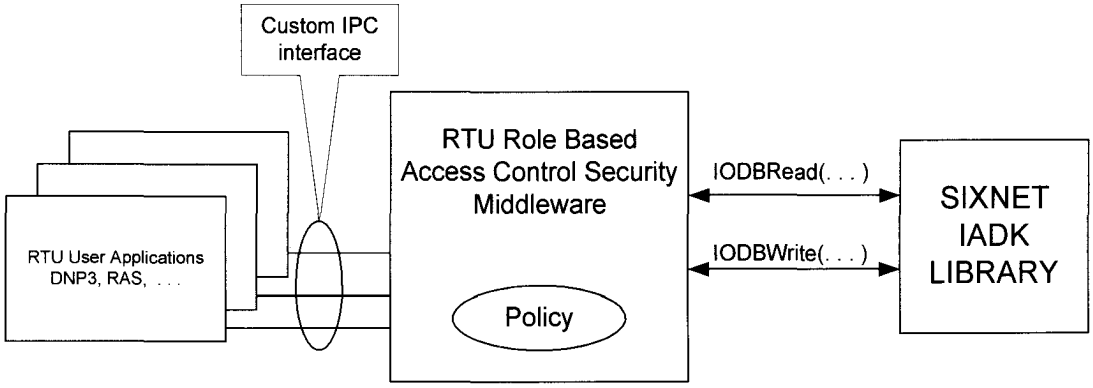


Figure 6.2. Prototype security middleware implementation.

The access control scheme was implemented as a middleware security layer using named and unnamed pipes. The module provides access to the protected RTU IO points by exposing an interface through which user programs can request processes IO operations. The interface is provided through the named pipe `/tmp/var/tmp/rtuIOserver`. The privileges of this pipe are set such that anyone can write to the named pipe but only its owner, the RTU middleware security layer can read from the pipe. The IADK calls IODBRead and IODBWrite require root privileges, which is given to the middleware security layer. Regular user processes can call IADK library functions, but they have no effect and the returned values are undefined. All hardened RTU user processes are executed without privileges, preventing them from making IO point changes except through the provided security middleware layer which implements the RTU role based access control.

The policy is stored in several permanent files in the flash memory of the prototype. The name of these files are: "USERS", "PERMS", "POINTS", "PAC", and "RAC". There are three fields in the USERS file: user id, user name, and user roles. Multiple roles are separated by a comma, and allow users to be assigned multiple roles. The POINTS file has three fields: logical id, logical index, and point type. The logical id and logical index uniquely name each point; for example, binary input one or analog output two. The point type is an element from point_type from the model described in chapter four, definition 4.5. The file PERMS has four fields, the operation, the logical id, the logical index, and the set of roles. This file creates the mapping between the IADK library routines, which represent all the possible permissions the prototype can execute, and the roles in the model. Multiple roles are separated by a comma. The constraints are stored in the two files: RAC and PAC, for role activation constraints and permission activation constraints respectively. There are four fields in RAC: a user id, a logical id and logical index (indicating the actual RTU point), a role, and a constraint set. Multiple elements in the constraint set are separated by a comma.

In each file, fields are separated by white space. New constraints can be added and existing constraints can be removed from the prototype by manually editing the files "RAC" and "PAC". The permissions for these files are set to require root privileges. For the prototype no additional interface to the constraint set was developed. It would be possible to create a custom remote and authenticated application to allow updating of constraints. However, DNP3 provides the ability to read and update files, and could be used to make policy changing. A more sophisticated constraint update interface could be built with some effort and should have little impact on the performance of the RTU.

A process makes a processes IO request by writing to the named pipe. The bytes are structured into a request message. Figure 6.3a shows the format of the request message and figure 6.3b shows the format of the response message. Before sending a request message a user program creates its own pipe, through which it will receive a response. The RTU access control enforcement module will open the pipe designated by *pipename* and write the response message back to the caller. The RTU access control enforcement module always sends a response message, even if the operation was not allowed, as not sending a response could potentially deadlock the user application. In the event that access is denied, this will be indicated in the return message's status flag. All this functionality could easily be wrapped into a library and hidden in a simple library call. But for initial implementation and testing it was left exposed.

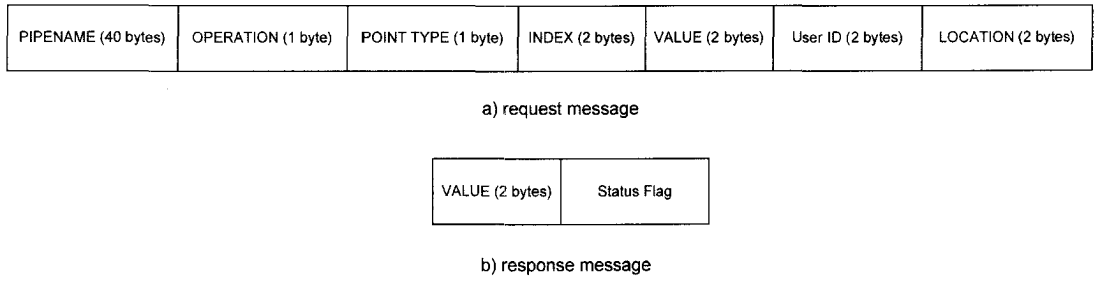


Figure 6.3. Security middleware request message format.

The request message includes an operation, a point type and an index. These indicate the operation to be carried out (read or write), the point type (analog input, analog output, digital input, digital output), and the point index (beginning at index 0) respectively. These fields are sufficient for the middleware layer to translate the request into an appropriate IADK. Upon receipt of a request the RTU Security Middleware (RMS) first consults the RTU access control policy to determine whether the operation is allowed or not by calling the check_access function, based on the algorithm defined in section four of chapter four.

Recall from chapter four that the authorization decision is based on the following: a users role, the permissions assigned to the users role, a set of context information, and the sets RAC and PAC. The focus of this dissertation has been authorization, not authentication, therefore in the prototype each message includes a *user id* that identifies the user associated with the request. This *user id* is provided by the security enhanced DNP3 described in the next section. Context functions provide the context information. *Time of day* and *day of the week* are easily provided by accessing the system clock. Time is always given in UTC [113] since RTUs might be located anywhere in the world, and accessed from a different time zone.

The location context function returns the location associated with a specific user, particularly the user associated with the current request. Since the user id information is supplied as field in the request, the location information is supplied as well. The request message format defines four locations based on definition 4.1. The network location or host id of the originating request is a logical approach to tying a user to a location. In the prototype, DNP3 link layer address is used by the DNP3 module to map a user to a location. DNP3 addresses are required to be unique, and though they are spoofable, the protocol layer security enhancements provided protection against such spoofing. The mapping used assigned DNP3 address is given in table 6.2

Table 6.2. DNP3 address to location assignment.

DNP3 Address	LOCATION
10	CONTROL_CENTER
11	PLANT_FLOOR
12	ENTERPRISE_CAMPUS
All other dnp3 address	UNKNOWN

The system state is the fourth and final context function. The internal representation of system state is based on definition 4.7. System state is determined by the internal state of the RTU, and is stored in a location accessible to the security middleware layer. The prototype begins in the start up state then transitions to the operating state. The system state could be changed by other processes (if they have permission). For testing purposes the system state can be manually manipulated by changing the current system state stored in the file `/etc/SYSTEM_STATE`.

6.3.3. Prototype development: SCADA access via DNP3 protocol

The final component of the security hardened RTU is the use of a security enhanced SCADA protocol. The security enhanced protocol described by Patel in his doctoral dissertation [29] was used as the basis for this portion of the hardened RTU prototype. The security enhancements were applied to the DNP3 protocol. A more detailed description of the DNP3 protocol is described in appendix B, and Patel's enhancements are fully explained in [29]. This section provides a brief description of the modifications related to the prototype implementation.

DNP3 is a SCADA protocol designed to allow devices to communicate and transfer data and control commands from one point to another. It supports both serial and TCP/IP communications with IP communications generally being achieved by tunneling the serial version inside TCP or UDP packets. In DNP3 the term *outstation* refers to devices or computers that are in the field and the term *master* refers to computers in the control center. The term *slave* is also used to refer to an outstation. DNP3 is a non-proprietary protocol and a full specification is available from www.dnp.org for a nominal fee.

Every DNP3 device has a database of different data types, analog inputs, analog outputs, binary inputs and binary outputs. They are organized as an array of values. Data items are identified by their data type and index, called a point index. The MTU or master uses the data values to display the state or condition of the physical system to which one or more outstations or RTUs are connected. The objective of the master is to keep its database updated and accomplishes this by sending requests (polling) to outstations. Outstations then provide the master with the value of data item or items that were requested. (Outstations may also send unsolicited data to masters in the form of an unsolicited response). Master may also send data to outstations, causing outstations to update local values, which will intern effect connected field equipment.

The DNP3 protocol is organized into layers that are similar to the standard OSI model. The top layer, or user layer, maps DNP3 data objects to local data. On master stations, the user layer initiates requests to outstations for data. On outstations, the user layer retrieves data from the outstation database in response to a master's request. The requested operations, the item or items on which the request is made, and any data need to complete the request are specified in the application layer message. The transport layer breaks up long application layer messages into smaller packets for the link layer and re-assembles them on the other side. The link layer makes the physical connection reliable. For the purpose of this discussion only the application layer is of concern since the developed security enhancements are applied at the application layer.

DNP3 application layer messages have three main components: an application header, a function code, and one or more DNP3 objects each of which consists of an object header followed by one or more object values. Figure 6.4 shows the general

structure of the DNP3 application layer fragments; masters send request fragments and outstations send response fragments. The application control octet contains an application layer sequence number and some status flags. The function code indicates the operation being performed, and the object header identifies the data point or points (range of indexes) on which the operation is to be carried out. The IIN in the application response fragment is used to communicate the internal status of the RTU or outstation to the master.

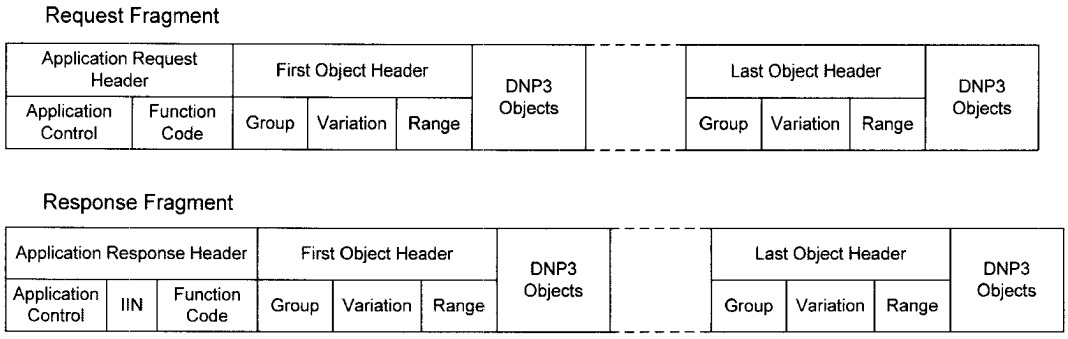


Figure 6.4. DNP3 application layer fragments.

In the challenge-response authentication security enhancement to DNP3 [29], a pre-shared secret is used to verify the authenticity of a communicating party and the integrity of a received message. It is assumed that the MTU and RTU have already established a pre-shared secret and the method for exchange is not specified here. Communicating parties, either the master or the slave (MTU or RTU) can, at any time initiate a challenge by sending a challenge message (a new DNP3 function). Upon receipt of a challenge message the outstation or master must reply with an appropriately constructed response message. Operation of either the slave or the master is suspended until an appropriate response message is received. After a predetermined and configurable number of consecutive failed responses, a master or slave terminates the connection. The challenge message includes a nonce value that is integral to the response

message and prevents replay attacks. The challenge message also specifies the cryptographic hash algorithm for the responder to use when building the response. The response message includes an HMAC [104;114] value. An HMAC is a keyed hash used for authentication. The HMAC value is computed over the nonce value and other predetermined data from the application layer fragment using the pre-shared secret as the HMAC key. A response is considered authentic if the received HMAC value matches the calculated HMAC value, as shown in figure 6.5. Typically the RTU would send a challenge when a connection request is received to prevent any further communication from proceeding without prior authorization.

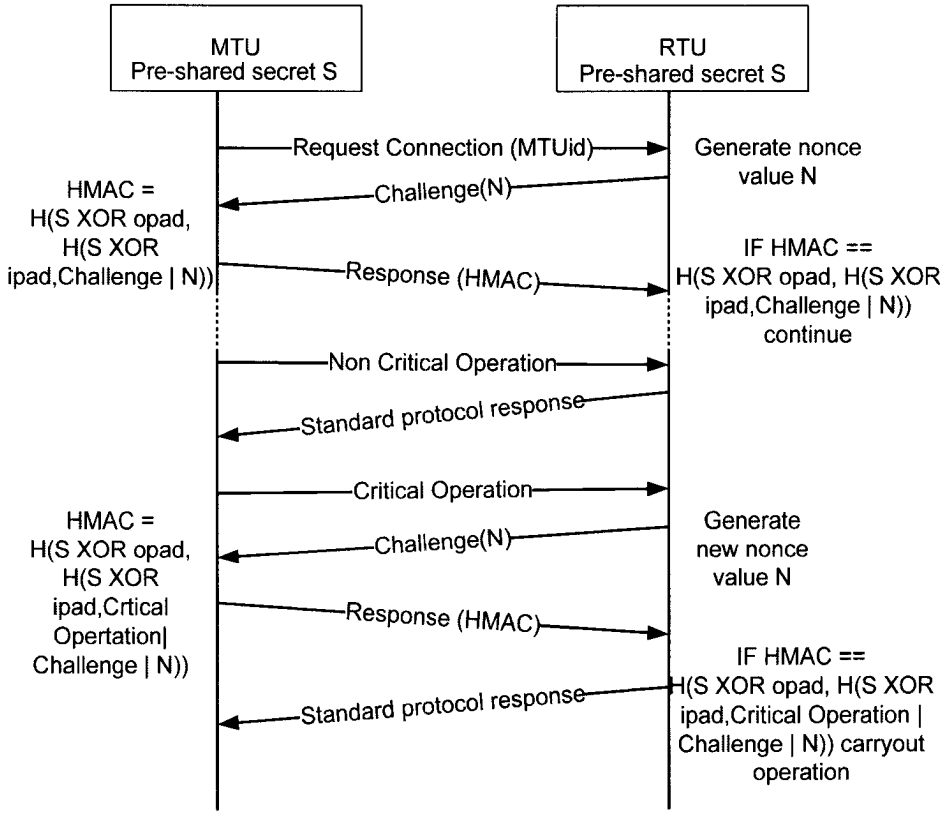


Figure 6.5. Challenge-response authentication for RTU – MTU communication.

During normal operation, an RTU (or MTU) receives requests for a variety of different operations, some of which may be considered more critical than others. Individual critical messages or operations can be explicitly authenticated using the challenge response scheme and including the application layer fragment containing the critical operation in the hash calculation. If an RTU (or MTU) receives a critical operation it issues a critical operation challenge that includes a nonce value and indicates to the sender the receiver is challenging the most recently received application layer message. The responder's HMAC calculation includes the nonce and the last application layer fragment sent to the challenger. Potential critical operations include: RTU output operations such as controls, set-point adjustments, and parameter settings and MTU receipt of atypical data or alarms.

The scheme as described by Patel is based on a single key and lacks the notion of a user. For the hardened RTU prototype implementation the scheme was extended to include the notion of a user. This was done by adding a user field to challenge-response messages, identifying the user providing the response. This requires each RTU, in this case the prototype to know each users key. HMAC calculation is done using the appropriate user key identified in the response message. Key distribution was beyond the scope of this work and was part of the initial setup of the RTU.

6.3 Hardened RTU prototype setup and configuration

Various components require parameters to fully describe their behavior, and these must be filled in during a setup phase. The following sections provide the instance specific assignments that relate to the deployed hardened RTU prototype. Where possible this information is given in tabular format for easier access.

6.3.1 RTU Role based access control policy

For RTU operation and testing to proceed an actual RTU access control policy had to be loaded in to the prototype RTU. The policy is simple but sufficient to exercise the model and carryout testing. The RTU access control policy is described by the following relations: UA, PA, PTC, PTA, RAC, PAC. In addition to these relations the set USERS needed to be populated. As mentioned in section 6.3.2, the specific RTU policy is stored in file form in the RTU's flash memory. Policy elements were added to the RTU using a standard editor to modify these files. The policy for the prototype is described in tables 6.3 through 6.6.

6.3.2 DNP3 configurations

There are several components of the DNP3 module that are configurable; the settings are given in table 6.7.

6.4 Test bed

To create a realistic testing environment the prototype RTU has been connected to a level control system in the process control lab of the chemical engineering department. The level control system is a simple process control setup and is ideal for the initial prototype testing since the failures can at worst overflow water onto the floor.

The level control system, shown in figure 6.6, consists of the following components:

- three bowl glass column
- a valve that controls the flow of water into the column based on air pressure input
- a manual screw valve at the bottom of the column that allows water to flow out of the column at varying rates
- a level control sensor that indicates the water level in the column

- an actuator that controls the air pressure applied to the water feed valve based on an applied voltage
- a flow sensor that indicates the amount of water flowing through the valve and into the column at a given time

The level sensor of the column is connected to analog input zero and the flow sensor output is connected to the analog input one. The valve position controller takes voltage between zero and nine volts, where zero volts closes the valve and nine volts opens the valve completely. The RTU analog outputs are 4-20 milliamps, so the digital outputs are used to generate the desired voltage. Binary outputs zero, one, and two, control the voltage level that is applied to the valve position actuator. If all three are off then zero volts are output and the valve is closed. If only one output is on then the generated voltage is approximately three volts, leading to a valve position of about 33%. If two outputs are on, then approximately six volts are generated leading to a valve being open about 66%. If all three digital outputs are on then the approximately nine volts are applied to the actuator and the valve is fully open.

Table 6.3 RTU users and role assignments

User Name	UserID	Roles
BOB	1	ENGINEER, OPERATOR
ALICE	2	OPERATOR
CHUCK	3	OPERATOR, ENGINEER
DORTHY	4	ADMINSTRATOR
EVAN	5	VENDOR
CC_DISPLAY	6	DISPLAY
CLOSED_LOOP_CONTROLLER	7	OPERATOR

Table 6.4. Permissions, point types and permission assignments.

Permissions (PERM)			
Operation	Point	Point Type	Roles
READ	ANALOGINPUT_0	STATUS	DISPLAY,OPERATOR,ENGINEER
READ	ANALOGINPUT_1	STATUS	OPERATOR,ENGINEER
READ	ANALOGINPUT_2	STATUS	ENTERPRISE
READ	ANALOGINPUT_3	STATUS	ENGINEER
READ	ANALOGINPUT_4	STATUS	ENGINEER
READ	ANALOGINPUT_5	STATUS	VENDOR
READ	ANALOGOUTPUT_0	CONTROL	DISPLAY,OPERATOR,ENGINEER
READ	ANALOGOUTPUT_1	CONFIG	OPERATOR,ENGINEER
WRITE	ANALOGOUTPUT_0	CONTROL	OPERATOR,ENGINEER
WRITE	ANALOGOUTPUT_1	CONFIG	ENGINEER
READ	BINARYINPUT_0	STATUS	OPERATOR,DISPLAY,ENGINEER
READ	BINARYINPUT_1	STATUS	OPERATOR,DISPLAY,ENGINEER
READ	BINARYINPUT_2	STATUS	OPERATOR,DISPLAY,ENGINEER
READ	BINARYINPUT_3	STATUS	OPERATOR,ENGINEER
READ	BINARYINPUT_4	STATUS	OPERATOR,ENGINEER
READ	BINARYINPUT_5	STATUS	VENDOR
READ	BINARYINPUT_6	STATUS	VENDOR
READ	BINARYINPUT_7	STATUS	VENDOR
READ	BINARYINPUT_8	STATUS	ENGINEER
READ	BINARYINPUT_9	STATUS	ENGINEER
READ	BINARYINPUT_10	STATUS	ENGINEER
READ	BINARYINPUT_11	STATUS	ENGINEER
READ	BINARYOUTPUT_0	CONTROL	ENGINEER,OPERATOR,DISPLAY
READ	BINARYOUTPUT_1	CONTROL	ENGINEER,OPERATOR,DISPLAY
READ	BINARYOUTPUT_2	CONTROL	ENGINEER,OPERATOR,DISPLAY
READ	BINARYOUTPUT_3	CONFIG	ENGINEER,OPERATOR,DISPLAY
WRITE	BINARYOUTPUT_0	CONTROL	ENGINEER,OPERATOR
WRITE	BINARYOUTPUT_1	CONTROL	ENGINEER,OPERATOR
WRITE	BINARYOUTPUT_2	CONTROL	ENGINEER,OPERATOR
WRITE	BINARYOUTPUT_3	CONFIG	ENGINEER
COLD_RESET	DEVICE	CONTROL	ENGINEER,OPERATOR

Table 6.5. Role activation constraints (RAC)

UserID	Role	Constraints
1	ENGINEER	OPERTATE_SECURE
5	VENDOR	00:00-10:00, 22:00-23:59
3	ENGINEER	UNKNOWN, ENTERPRISE
7	OPERATOR	UNKNOWN,ENTERPRISE,PLANT_FLOOR

Table 6.6. Permission activation constraints (PAC)

Role	Permission	Constraints
OPERATOR	WRITE BINARYOUT_0	UNKNOWN
OPERATOR	WRITE BINARYOUT_1	UNKNOWN
OPERATOR	WRITE BINARYOUT_2	UNKNOWN
ENGINEER	WRITE ANALOGOOUT_1	OPERATE_SECURE

Table 6.7. DNP3 settings

Setting	Value	Description
Periodic Challenge timeout	2 minutes	The time between random challenge messages
Response time out	5 seconds	The time an outstation or MTU will wait for a response before assuming it is lost.
Critical functions	WRITE, SELECT, OPERATE, COLD_RESTART	Use of these functions causes the RTU to send a challenged even if the authentication period has not expired.
HMAC hash	SHA-1	The hash algorithm used in calculating the HMAC
IP Address	xxx.xxx.49.249	Internal LAN IP address of RTU
DNP UDP Port	20000	UDP port DNP3 server listens on

6.5 Hardened RTU prototype testing

Testing of the hardened RTU prototype was divided into two different categories; performance and security. Both performance testing and security testing were conducted using the test bed previously described. To interrogate the hardened RTU prototype, several MTU and HMI programs were used. These programs supported the security enhanced protocol used by the prototype, and provided a simple interface that allows a user to issue commands to the RTU, and see the results displayed on the screen. Section 6.6 describes the performance testing and results, and section 6.7 described the security testing and results.

6.6 Performance Testing

As discussed in chapter two, SCADA systems have different performance requirements than do traditional IT systems. Though not all SCADA systems and process control systems have hard real time requirements, it is important that the SCADA system (in this case the RTU) have reasonably short response times. Since different systems have different requirements, there is no established targeted response time.

Furthermore, in evaluating the performance of impact of the AGA encryption scheme for serial communications [115] found that while operators perceived polling as continuous, there are generally small delays built into SCADA systems that can absorb some additional response overhead resulting from the addition of security. Some performance testing of the prototype was conducted to access the impact of security measures on the prototypes response to SCADA message requests.

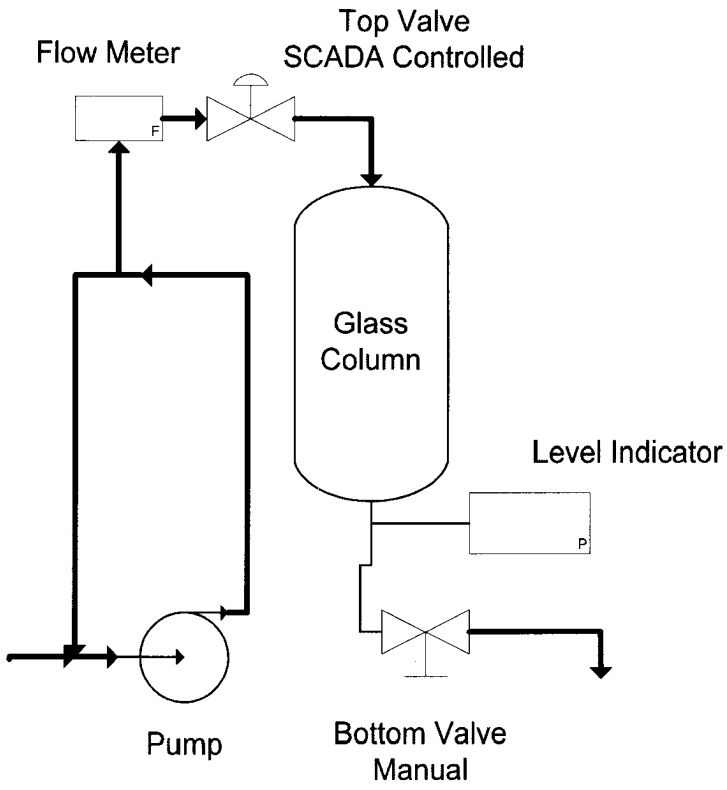


Figure 6.6. Diagram of the level control system in the process control lab.

As DNP3 was the chosen SCADA interface to the RTU, performance measured the prototype RTU's response to DNP3 requests. While response times of a single DNP3 request are important, more complicated actions are also of interest. Therefore several different tasks or workloads were used during performance testing. Each task was

conducted using an appropriate user from the list of users in table 6.2. Response time is defined in equation (6.1).

(6.1) $R = t_{response} - t_{request}$.

Where $t_{response}$ is the time at which the final DNP3 response was received (or in the event that there was a loss of communication, that the final response timer timed out) and $t_{request}$ is the time at which the first DNP3 request was initiated by the control program.

The response time R, includes any latencies introduced by security measures as well as latencies introduced by the amount of network traffic. To reduce the impact of specific network traffic conditions, each task was continuously repeated for one hour. The overhead of control calculations in the closed loop control task were measured independently and found to be between zero and three microseconds. This overhead is left in the performance results, since it is several orders of magnitude smaller than the response times. The MTU program, which supports the master DNP3 protocol, was instrumented to record time elapsed in milliseconds while spent in one loop iteration. Each task was then implemented inside the loop, and timing gathered by the program indicate the amount of time it took to complete one round of a specific task. A brief description of each task is given in section 6.6.1.

6.6.1 Performance testing task descriptions

Simple Read

Reading individual points from the RTU is a basic SCADA task. This task consists of reading an individual outstation point, specifically analog input one which is the flow reading. Reads make up a significant portion of SCADA activities, and so this

task serves a unit operation base line for reads. The task is conducted as the user BOB, who is assigned to the engineering role.

Simple Write

Making changes to interval points is another basic SCADA task. The simple write task writes a value to the prototype, specifically it writes a value to analog output one. Analog output one represents a virtual IO point influencing how the system reports the column level. This task is also carried out by the user BOB.

Static Data Poll

This task sends the RTU a READ function code followed by a special DNP3 object that indicates to the RTU to send the value of every outstation point. These values are sent in single DNP3 response. For each point the RTU role based access control layer must decide whether the user has permission to read the point. Confidentiality is not a primary RTU concern, but the RTU access control model supports defined access in such a way that there could exist outstation points, that are not readable by all users. In fact the policy developed for testing includes this situation. If access is denied, the RTU access control layer returns a status indicator to a calling module indicating this. The DNP3 module treats this as an offline condition. This task is conducted by the CC_Display user. This user has limited privileges and is intended to be used by display consoles, that provide status information, but not process control.

Closed_Loop_Control

This task represents a PID control loop block that might be found in a process control setting. The task consists of reading the level of the water in the column from analog input zero, making simple PID control calculation, and then writing the control calculation result to analog output 0. Analog output zero is a 4-20 mA output that could be used to control the valve position, though in the test setup this was not possible since

the valve position control uses voltage as the control signal and not current. This task is carried out by the user CLOSED_LOOP_CONTROLLER. This user is intended to represent a host in the control center dedicated to carrying out closed loop process control. The user is assigned to the *operator* role, and further restricted by a RAC that prevents that user id (which may have weaker authentication, being a machine account), from activating the role from any location other than the control center.

Complex Operation

This task represents a complex, multi-step, operation, such as might be carried out by an operator, engineer or vendor. It involves reading values, and then making some adjustment to the binary outputs using a control block and select before operate. This task serves to evaluate a human in the loop control, which is typical of many SCADA systems. The task consists of reading the water level in the column, by reading analog input zero, and then use select and operate on binary output zero then binary output one then binary output two to effectively set the actual valve position. The task is carried out by the user Alice, who is assigned to the operator role.

6.6.2 Performance Results

Each task was allowed to run in a continuous loop for one hour. Then log files of the timing data and DNP3 statistics were collected and analyzed. Tables 6.8 – 6.12 present the results from each test, and table 6.13 summarizes the response time results of all tests. The mode of the recorded response times for each test was chosen to represent the average response time. Since each test experienced some response time outs, in which for some reason a response to a request was never received, some response times for each task were actually the response timeout times. Reporting mode as opposed to

mean prevents these times from inappropriately influencing the results. The simple read had the shortest response time of 49 milliseconds and the complex operation had the longest response time, over two seconds. This is what should be expected, given the number of critical operations included in each complex operation. The static data poll was the only other task besides simple read that included no critical functions. Its higher response time reflects the extra work done by the access control layer in determining access for each outstation point. However the found response time is better than reading all points individually as the total number of outstation points is twenty six. Twenty six reads at 50 milliseconds each, would result in a total response time of 1300 milliseconds, substantially longer than the reported 379 milliseconds.

Table 6.8. Performance statistics for the simple read task.

Simple Read	
Average Response time	49 milliseconds
Transmitted Read Requests	41494
Transmitted Write Requests	0
Transmitted Selects	0
Transmitted Operates	0
Number of response time outs	58
Challenges Sent	30
Challenged Received	30

Table 6.9. Performance statistics for the simple write task.

Simple write	
Average time to receive a response	399 milliseconds
Transmitted Read Requests	0
Transmitted Write Requests	7691
Transmitted Selects	0
Transmitted Operates	0
Number of response time outs	51
Challenges Sent	30
Challenged Received	7658

Table 6.10. Performance statistics for the static data poll task.

Static Data Poll	
Average time to complete one poll	379 milliseconds
Transmitted Read Requests	597
Transmitted Write Requests	0
Transmitted Selects	0
Transmitted Operates	0
Number of response time outs	48
Challenges Sent	29
Challenged Received	30

Table 6.11. Performance statistics for the closed loop control task.

Closed loop control	
Average time to complete one loop	499 milliseconds
Transmitted Read Requests	6589
Transmitted Write Requests	6589
Transmitted Selects	0
Transmitted Operates	0
Number of response time outs	68
Challenges Sent	30
Challenged Received	6581

Table 6.12. Performance statistics for the complex operation task.

Complex operation	
Response time to complete the entire operation	2506 milliseconds
Transmitted Read Requests	1267
Transmitted Write Requests	0
Transmitted Selects	3801
Transmitted Operates	3762
Number of response time outs	75
Challenges Sent	30
Challenged Received	7532

Table 6.13. Summary of response times for each performance task.

Task	Response time in milliseconds
Simple Read (read one analog input point)	49
Simple Write (write one analog output point)	399
Static Data Poll (read all RTU points)	379
Closed Loop (read 1 point write 1 point)	499
Complex Operation (read 1 point, select / operate 3 points)	2506

6.7 Security Testing

The second component of prototype testing focused on security. The goal of the hardened RTU is to increase security for RTUs. Both the reduced Linux kernel and the RTU role based access control layer were developed for the hardened RTU to enhance security. The reduced Linux kernel was intended to reduce security threats resulting from vulnerabilities in COTS operating systems by reducing the size of the TCB and the number of lines of code compiled in the kernel. Evaluation of this approach was discussed in the development of the prototype in section 6.3. It was shown there that the kernel was significantly reduced in size and it was argued that an inference could be made that an equivalent reduction in vulnerabilities was achieved. Further evaluation of testing of this mechanism was achieved using the standard IT penetration testing tools: NMAP, NESSUS, and fuzzball. These tools were used against the prototype RTU while it was connected to the level control system and providing an interface to a human controller operating a console based control application.

The RTU role based access control layer was intended primarily to mitigate insider threat attacks, and to a lesser degree place limits on other software components. Security testing of this hardened RTU prototype component was conducted by carrying out simulated insider attacks against the hardened RTU prototype while it was operating and connected to the level control system and providing SCADA to a human operator operating the level control system from a console based SCADA application. Sections 6.7.1 through 6.7.4 describe the security tests conducted, the results and any related discussion.

6.7.1 NMAP Scan

Two nmap scans were done, a TCP scan and a UDP scan. The output of the two scans is shown in figure 6.7. The nmap scan identified ftp and telent ports as open. The prototype does not have a serial console interface, therefore these services were needed on the prototype to allow development access during testing. However a custom script was tied to Internet daemon inetd to activate these services. That script reads the value of binary input eleven. If the input is one then telent or ftp is started by the script. If the value of binary input eleven (which is tied to physical switch in the development platform), is zero then ftp or telnet are not started. So while nmap identified these as open, actually using the services was reserved for development aspects, not deployment, and is controlled by a physical switch.

6.7.2. Nessus scan

Following the nmap scan a nessus vulnerability scan was directed at the hardened RTU. The result of this scan correlates with the nmap scan. The complete nessus report is listed in appendix D. Nessus identified ftp and telnet as open, but protected with tcpwrapper. As mentioned in section 6.7.3, telnet and ftp were disabled using a custom script to allow a hardware switch to enable them if needed. This setup allows inetd to answer requests for which it is configured, and this is the reason Nessus identified them as protected by tcpwrapper. Nessus also correctly identified the prototype as running the Linux 2.4 kernel.

```
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2007-10-30
10:43 EDT
Interesting ports on private049249.private.louisville.edu
(10.165.49.249):
Not shown: 3165 closed ports
PORT   STATE SERVICE    VERSION
21/tcp open  tcpwrapped
23/tcp open  tcpwrapped
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux 2.4.0 - 2.5.20
Uptime 2.900 days (since Sat Oct 27 13:32:48 2007)

Nmap finished: 1 IP address (1 host up) scanned in 1515.371 seconds
```

(a) nmap –sT –sV -0 xxx.xxx.49.249

```
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2007-10-30
13:08 EDT
Interesting ports on private049249.private.louisville.edu
(10.165.49.249):
Not shown: 510 closed ports
PORT      STATE         SERVICE
20000/udp open|filtered unknown

Nmap finished: 1 IP address (1 host up) scanned in 511.571 seconds
```

(b) nmap –sU –p1-500,19995-20005 xxx.xxx.49.249

Figure 6.7 Output from the nmap scans of the prototype hardened RTU.

Nessus also identified the hardened RTU as vulnerable to the 'nestea' attack, CVE: CAN-1999-0257, and claimed it was possible to make the server crash using this attack. The 'nestea' attack is a type of Denial of Service (DoS) similar to a teardrop attack. The 'nestea' attack is an off by one IP header bug. Recovery from the attack usually consists of rebooting the server. However, for RTUs, without a functioning network interface, rebooting may not be a practical solution. According to Nessus, the hardened RTU could be crashed using this remote DoS exploit. Exploit code that carries out a nestea attack was downloaded from the web, and compiled on the attack machine. The command

```
nestea xxx.xxx.67.47 xxx.xxx.49.249 -n 1000
```

was executed from the attack machine to launch 1000 'nestea' attacks against the hardened RTU prototype. The hardened RTU prototype remained running and continued to provide level control readings, and allowed the operator to make changes to the valve position. It was then concluded that the hardened RTU was in fact not vulnerable to the 'nestea' attack. Nessus did not identify any other vulnerabilities, though it did point out that packets with the syn flag set were not dropped.

6.7.3 Fuzzball

As a final general security test, the open source fuzzier fuzzball was used. The hardened RTU prototype exposes on the DNP3 network service, but parts of the TCP/IP stack are exercised as well. Fuzzball was used to send packets to the hardened RTU prototype with non standard IP and TCP settings. These packets can activate flaws in the TCP/IP stack that might not otherwise be activated by more normal packets. Fuzzball was run against the prototype while it was operating and providing status and control to a human operator. During and following the fuzzball test the prototype hardened RTU continued to function normally.

6.7.4. Insider attacks

A primary focus of the RTU's novel security, the new RTU role based access control model, was to enable fine grained access control while still mitigating potential insider attacks. Therefore a major component of the security testing focused on simulating a variety of insider based attacks and confirming that the RTU role based access control system and enforcement algorithm did indeed provide protection. For this testing component, it was important users be associated with a location. The mapping

between location and DNP3 address was given in section 6.2.1. The following insider based attacks were carried out against the prototype hardened RTU.

Insider attack scenario one: The user ALICE, from the control room writes the value 30 to analog ANALOGOUTPUT_1. ANALOGOUTPUT_1 is a dead band value that indicated the percent change in the column level required to cause the RTU to generate an unsolicited response. Increasing this value would mean that a greater change in the column level was required before the RTU would issue an unsolicited response. Since ALICE is assigned to the OPERATOR role and permission to WRITE ANALOGOUTPUT_1 is not assigned to the role OPERATOR, ALICE is prevented from carrying out this attack by the role assignment relation. The prototype correctly blocked this action.

Insider attack scenario two: The user ALICE SELECTS and then OPERATES BINARYOUPUT_1 on from an unknown host. This will increase the openness of the valve allowing water into the column at a greater rate. Operators are only allowed to control the valve position from the control room. ALICE is granted permission to operate BINARYOUPUT_1 by her membership in the OPERATOR role, but is prevented from activating the permission by a the permission activation constraint <OPERATOR, WRITE BINARYOUTPUT_1, {UNKNOWN}>. The prototype successfully blocked this operation.

Insider attack scenario three: EVAN sends SELECT and OPEATE BINARYOUTPUT_0 on, SELECT and OPERATE BINARYOUTPUT_1 on, and SELECT and OPERATE BINARYOUTPUT_2 on. This will fully open the valve that controls the flow of water into the column. Vendors, who are essentially external

engineers, need access to the RTU from time to time, but are not permitted to control physical processes. EVEN as a member of the VENDOR role is not granted permission to write values to BINARYOUPUT_0, BINARYOUTPUT_1, and BINARYOUTPUT_2. The prototype successfully blocked this operation.

Insider attack scenario four: EVAN sends READ ANALOGINPUT_5, READ BINARYINPUT_5, READ BINARYINPUT_6, READ BINARYINPUT_7 at 8:00 UTC. ANALOGINPUT_5, BINARYINPUT_5, BINARYINPUT_6, and BINARYINPUT_7 are some type of status value. Vendors are allowed to read these values, and the necessary permissions are available to the user EVAN through the assignment to the VENDOR role. However EVAN's work schedule is known and established to be from 10:00 – 22:00 UTC. Therefore EVAN is prevented from activating his VENDOR role assignment by the RAC <5, VENDOR, {00:00-10:00,22:00-23:59}>, and therefore cannot access the permissions assigned to VENDOR. The prototype successfully blocked this operation.

Insider attack scenario five: The hardened RTU prototype state was first set to OPERATE_SECURE. The secure operating mode might be triggered by external intrusion detection system (IDS) event or a local event. After the RTU state had changed, BOB writes the value five to ANALOGOUTPUT_1. ANALOGOUT_1 is an engineering dead band value that configured when the RTU sends unsolicited responses concerning the column level. BOB has been granted this permission through the ENGINEERING role. However, the PAC <ENGINEER, WRITE ANALOGOUTPUT_1, {OPERATE_SECURE}> prevents the permission from being

activated through the ENGINEERING role when the RTU is in secure operating mode. The prototype successfully blocked this operation.

Table 6.14. Insider attack scenarios

	Attack Conditions	Attack stopped because
Scenario 1	ALICE $\in$ *USERS* roles(ALICE) = OPERATOR location(ALICE) = CONTROL_ROOM op = write point = ANALOGOUTPUT_1	<<write, ANALOGOUTPUT_1>,OPERATOR > $\notin$ RA
Scenario 2	ALICE $\in$ *USERS* Roles(ALICE) = OPERATOR Location(ALICE) = UNKNOWN op = write point = BINARYOUTPUT_1	$\exists$ *(cr,cp,cs)* $\in$ *PAC*| *OPERATOR* = *cr* $\wedge$ (write BINARYOUTPUT_1)= *cp* $\wedge$ *UNKNOWN* $\in$ *cs*
Scenario 3	EVAN $\in$ *USERS* *roles(EVAN) = VENDOR* *op = write* *point = BINARYOUTPUT_1,* *BINARYOUTPUT_2,* *BINARYOUTPUT_3*	<<write, BINARYOUTPUT_1>,VENDOR > $\notin$ RA, <<write, BINARYOUTPUT_2>,VENDOR > $\notin$ RA, <<write, BINARYOUTPUT_3>,VENDOR > $\notin$ RA,
Scenario 4	EVAN $\in$ *USERS* *roles(EVAN) = VENDOR* *time_of_day=08:00* *op = read* *point = ANALOGINPUT_5,* *BINARYINPUT_5,* *BINARYINPUT_6,* *BINARYINPUT_7*	$\exists$ *(cu,cr,cs)* $\in$ *RAC*| *EVAN* = *cu* $\wedge$ *VENDOR* = *cp* $\wedge$ *08:00* $\in$ *cs*
Scenario 5	system_state() = OPERATE SECURE BOB $\in$ *USERS; op = write; point =* *ANALOGOUTPUT_1*	$\exists$ *(cr,cp,cs)* $\in$ *PAC*| *ENGINEER* = *cr* $\wedge$ (write ANALOGOUTPUT_1)= *cp* $\wedge$ *OPERATE_SECURE* $\in$ *cs*

Another key security testing result was affirmation that for COTS systems, high assurance middleware layer security can be difficult to achieve. In implementing the prototype RTU, interprocess communication was used to achieve cooperation among RTU components and form the basis for the middleware layer security component, not an unusual or uncommon approach. A serious problem with this approach, at least in the Linux IPC functionality, is the lack of identification of the calling process. Ideally the

security layer would perform authentication, but the anonymity of IPC in the Linux kernel make that difficult and in the prototype, authentication was achieve using the security enhanced DNP3 protocol. A means of transferring authentication from one entity to another is needed. This could be achieved by third party authentication server process, which could then serve as the authentication authority for all processes. However, this functionality is difficult to provide in Linux because there is no simple, high-assurance, way for the receiver of a message to know the true identity of a sender. This is one area where microkernels show their benefits, as their IPC is not anonymous; this is explored and explained in more detail in chapter seven.

6.8 Conclusions

This chapter has presented the development, implementation, and testing of a security hardened RTU prototype. The prototype was developed using the open mIPM from SIXNET, an actual RTU platform that support OEMs. The hardened RTU prototype included a custom compiled reduced Linux kernel, security enhanced SCADA communications running over UDP, and the RTU role based access control developed in chapter four. A test environment was created for the RTU using a level control system made available by the Department of Chemical Engineering at the University of Louisville. Performance and security testing of the security hardened RTU was then performed. Performance testing included simulating various tasks involved in monitoring or operating the level control system using the prototype hardened RTU. All but one task was completed by the prototype RTU in less than 500 milliseconds. Security testing including: basic scanning using nmap and nessus, launching the 'nestea' remote exploit (identified by nessus) against the RTU, and simulating five insider attacks. The

hardened RTU continued to operate during and after the scans and the remote exploit attack, and the insider attacks were successfully blocked by the RTU role based access control security middleware. The attacks carried out for this testing were limited to trying to change only one or a handful of points, due in part to the limited complexity of the test bed. While more complex attacks that try to change many RTU points are possible, the current test bed is not sufficient to fully test such attacks. However, vulnerability to such tasks would more likely result from a misconfiguration issue than from a failure of the model to provide sufficient protection.

CHAPTER VII

MICROKERNELS FOR HARDENED RTUS

Chapter six discussed a hardened RTU prototype using a reduced Linux kernel and the RTU role based access control scheme described in chapter four. Chapter five presented two approaches for minimal kernel RTUs, the reduced COTS kernel approach used in the prototype development in chapter six, and microkernels. The reduced COTS kernel approach was used in the initial prototype development to allow quicker development of a functioning prototype and use of the RTU role based access control security middleware. This chapter further investigates the use of a microkernel in developing a hardened RTU.

As discussed in chapter two, the microkernel idea originated with Brian Hansen's Nucleus [88] and gained popularity with MACH [92]. Initial poor IPC performance and the assertion that the poor performance was inherent to microkernels led to this approach falling out of favor for many years. However, thanks mostly to Leidke [87;116], there has been resurgence in microkernel research and development as well as increased interest in their commercial use. The MILS architecture [99;100] is a developing standard for high-assurance systems, and is based on a type of microkernel called a separation kernel. While a MILS system, suggested as potential platform for security hardened RTU in chapter five, was not available for evaluation at time of this dissertation, there were some microkernels available for investigation. The following

microkernels were considered as possible candidates for use in a security hardened RTU: QNX, VxWorks, Mach, Minix, and L4.

Two criteria were considered in determining a potential microkernel implementation for investigation. The first criterion was that the microkernel be open source. There are several reasons for the open source requirement. First, as discussed in chapter two, SCADA vendors and operators are strongly motivated to contain costs. Open source systems alleviate the licensing cost associated proprietary operating systems. Of equal value is the availability of source code. Since SCADA devices can potentially have a long life time (fifteen to twenty years) the availability of source code assures that the product can be supported throughout its lifetime. An open source microkernel also assures vendors or product developers that the source code will be available throughout a project, and avoids the potential problems that can occur when commercial systems are acquired by another company or become unavailable due to bankruptcy. Finally, an open source microkernel allows continued development of this project and the possibility of sharing results with industry practitioners. The second criterion for consideration was that the microkernel provide partitioning, in both space and time, and support some type of RT scheduler. This is needed to allow the isolation of RTU components described in chapter five, and allow the RTU to achieve real-time requirements.

Of the identified potential microkernels, QNX[1] and VxWorks were eliminated because they are not open source. Mach was eliminated because it is a first generation

[1] QNX has very recently released part of their system under an open source licencing agreement. However, there was not time to sufficiently investigate this recent addition to the open source microkernel community. QNX is a real-time operating system the focus of which has therefore been performance and not security.

microkernel known to have poor performance characteristics, and a large code base. Minix was eliminated because it lacks a real-time scheduler and supports only IA32 processors. This leaves the L4 microkernel as the choice for further investigation. Fortunately there are several open source L4 implementations available that could be used in a security hardened RTU. A hardware platform and L4 implementation were selected for prototype development and evaluation. The hardware platform and L4 implementation were obtained and the final section of this chapter presents the results of some initial experimental work to investigate using OKL4 microkernel as the basis for a security hardened RTU, and to determine the IPC overhead inflicted on the hardened RTU.

7.1 The L4 Microkernel

The L4 microkernel, originally developed by Liedtke, adheres to Liedtke's microkernel design criteria of allowing only those features into the kernel, which cannot be exported out of the kernel. L4 is not an operating system, but is rather a minimal base on which a complete operating system can be built. The most fundamental task of an operating system, and therefore of a microkernel, is to provide abstractions for sharing resources securely. Towards this end, and in maintaining minimality with respect to the kernel, L4 provides only a few basic abstractions and mechanisms.

Address spaces

Data, other than hardware registers, which are accessible to a thread are contained in the threads *address space*. *Address spaces* are L4's basis for protection. An *address space* in L4 is a partial mapping from virtual memory to physical memory. Threads can share data by mapping parts of their *address space* to other *address spaces*. The mapping

can be revoked by the *mapper* at any time. In addition to mapping, a thread can grant parts of address space to another address space. In this case the *grantor* gives up control of the data and can no longer access that part of the virtual address space. The *grantee* receives full control of that data and may subsequently map or grant it to other address spaces. Through the use of mapping and granting described above, L4 *address spaces* can be recursively constructed. The concept of a *task* is basically synonymous with address space, where a task is a set of threads that share an address space.

Threads

Threads are the basic unit of execution in L4. Every thread is "attached" to an address space, which it may share with other threads. A thread has a unique identifier (UID) and a register set that includes an instruction pointer and a stack pointer. Threads communicate with each other through interprocess communication (IPC) primitives provided by L4. Threads can also communicate with each other using shared memory. Threads within an address space can access shared memory, and a thread can also map memory into other thread's address space and share memory that way.

IPC

The heart of L4 is the message-passing interprocess communication (IPC) that it supports. L4 IPC is synchronous and unbuffered. L4 IPC can be used to pass data by value or by reference (using mapping or granting). Since IPC is synchronous L4 IPC can also be used for thread synchronization. L4 supports the following basic IPC primitives:

- receive – wait for a message from a specific thread
- reply_wait – send a reply message to a client thread, and wait for the next request
- send – send a message to a thread
- wait – wait for a message from any thread

UID

A UID is a unique identifier for threads in an L4 system. A thread's UID is composed of the task number of the thread's address space and the thread's local thread number within that task. The UID is used by kernel IPC calls.

An L4 system is then composed of address spaces populated by threads executing code in their address space. Figure 7.1 depicts an L4 based system. Individual threads, indicated by a thread UID, operate on data stored within their address space. The L4 kernel assures that threads are not able to execute instructions in other address spaces or access data in other address spaces. However, in L4 memory is not managed by the kernel. Instead, L4 supports what are called external pagers, which are user level threads that manage memory. An initial task, referred to in L4 as Sigma0, is privileged in that it is run first by the kernel, and claims all the physical memory of the system. Sigma0 can then map or grant that memory to other tasks. Other tasks can then map or grant that memory to a succession of other tasks, allowing for the recursive construction of address spaces. Pagefaults in L4 are translated by the L4 kernel into an IPC message to a tasks pager. Initially Sigma0 would be the pager for all tasks, but through the construction of recursive address spaces, there can then be multiple pagers in a given system.

Threads are scheduled by L4 according to three parameters: time slice length, thread priority, and maximum controlled priority. Each L4 thread has a time slice value that indicates the amount of time for which that thread will be scheduled, and is stored as part of the threads control block. Different threads can have different time slice values. When a running thread uses up its allotted time slice the scheduler selects and starts the next run-able thread. The L4 kernel supports 256 different priority levels, 0 – 255, with

255 having the highest priority. L4's scheduler has a, possibly empty, queue for each priority level. All the queues together form the ready queue, and a thread's priority level indicates to which queue the thread is assigned. A thread's priority can be changed, and will affect the queue to which it is assigned. The maximum controlled priority is a task level value that prevents the threads within a task from being assigned a priority higher than the maximum controlled priority for that task. This limitation is also applied to newly created threads as well.

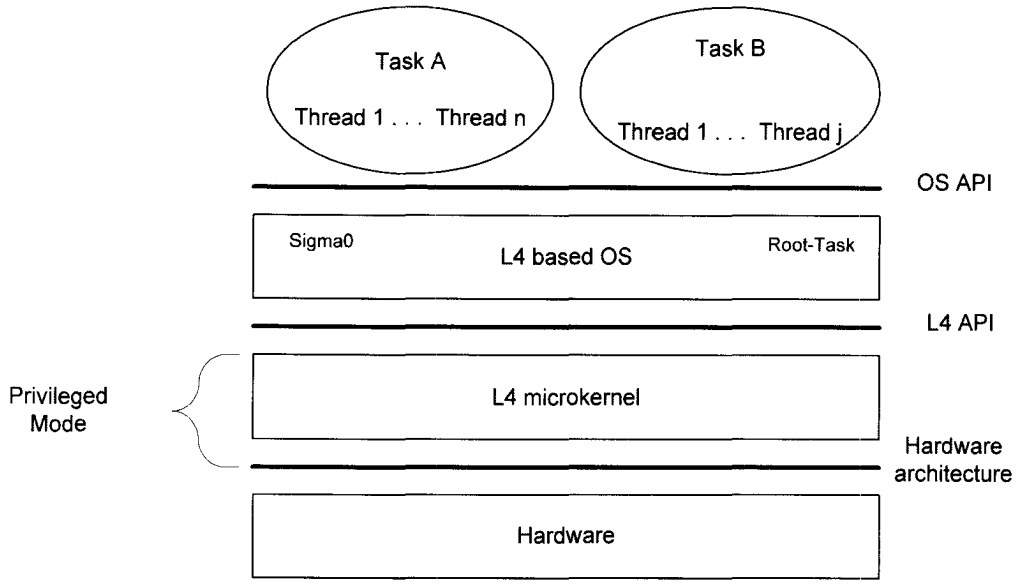


Figure 7.1. Structure of an L4 based OS system.

With respect to the security objectives for the hardened RTU, specifically the space and time partitioning, L4 provides both space and time partitioning. L4's address space abstraction provides a mechanism to achieve space partitioning. In addition L4 scheduling, through time slices and priorities, supports real-time scheduling. Along with time and space partitioning the microkernel must support a way for isolated components to cooperate. L4 supports thread cooperation through a well defined and supposedly fast IPC system call interface mentioned previously. These attributes together make L4 an

excellent choice as the platform for a security hardened microkernel based RTU. They allow the primary goals identified in chapter five: strong isolation of RTU components and controlled interaction along a well defined interface.

7.1.1 L4 implementations

There have been a number of L4 implementations over the years. Jochen Liedtke's initial implementation of L4 called L4/x86 was written for x86 machines, and is no longer supported. It served as the basis for performance evaluations that demonstrated L4 could achieve reasonable IPC performance needed to build a system with overall acceptable performance. However that implementation is no longer supported or developed. Since then several groups have developed their own L4 implementations and the L4 API has evolved. The early L4 API is referred to as version 2 (V2) and the more recent API is referred to as version 4 (V4). Three L4 implementations were considered as potential candidates for investigation: Pistachio, Fiasco, and OKL4. A brief description of each is given in the following paragraphs.

Pistachio [117]: Pistachio is the most recent microkernel developed by the System Architecture Group at the University of Karlsruhe. Pistachio implements the L4 version 4 API and is written in C++. Pistachio includes Sigma0 and Sigma1 which server as pagers for the entire L4 system. Pistachio is fully 32 and 64 bit clean and provide support for multiprocessors. Pistachio supports Alpha, AMD64, ARM, IA32, IA64, MIPS and PowerPC processors. Pistachio is actively maintained and is licensed under a BSD license.

Fiasco [118]: Fiasco was developed at Technical University Dresden (TUD). Fiasco is licensed under GPL, and is freely redistributable. Fiasco implements the L4v2

ABI which was the original L4 specification defined by Liedtke. Fiasco is part of the TUD DROPS operating system. Fiasco is implemented in C++ and is currently supported on i486 and later Intel architectures, ARM processors SA-1100 and XScale PXA 25x. Fiasco is still under active development.

OKL4 [119]: OKL4 descends from a Pistachio-embedded system which was developed by National Information and Communication Technology Australia (NICTA). OKL4 is now developed and maintained by OK-labs [120]. The kernel supports the L4 version 2 API and is written in C++. The OKL4 release supports ARM, x86, and MIPS processors, and is targeted toward embedded systems. OKL4 is released under the BSD license. There is also a commercial licensing available, and OKL4 has actually been used in a least one commercial device, the OpenMoko phone [121].

Each of these implementations was downloaded and the basic system built using the provided build system. Pistachio uses the *Scons* build system, Fiasco uses GNU Make, and OKL4 uses its own python based system that is provided in the download and is completely self-contained. This initial build was used to assure that the implementation was reasonably stable for carrying out additional evaluation. Each of these systems supports at least one kind of simulation. Fiasco has a user mode fiasco, FiascoUX, that enable the kernel and other components to be loaded and run as a standard Linux process, in much the same way that user mode Linux (UML) works. Pistachio uses *Qemu* and OkL4 uses *Skyeye* or *Qemu* to emulate hardware. The standard build of each of the systems was successfully loaded and executed in the appropriate simulator. However, though the availability of simulators makes it possible to develop and test prototype implementation code, for the hardened RTU, performance on real

hardware is a key issue of this evaluation. Therefore, before making a final decision on an L4 implementation a hardware platform was selected. The selection criteria and section process are described in the following section.

7.2 Development Platforms

There are many available development platforms from which to choose. The list below describes the basic requirements that the development platform needed to meet:

1. Small embeddable computer with sufficient resources to store and load the microkernel and eventually support network communications, and analog and digital IO,
2. Ability to attach external digital and/or analog IO circuitry,
3. Support potential future commercialization activities,
4. Full console access to a flexible boot loader, preferably Uboot,
5. Supported by a GNU tool chain,
6. Use a processor supported by at least one of the L4 implementations listed in section 7.1.1.

7.2.1 Platform Analysis

This section presents a brief overview of the development platforms considered.

SIXNET mIPM

The SIXNET mIPM, used for prototype development in chapter six, was considered as potential and desirable development platform. To load the microkernel, root-task, and any other tasks requires a flexible boot loader. The mIPM uses Uboot, but a command line interface to the boot loader is not accessible. The mIPM Uboot bootloader is configured to boot a Linux kernel, located in a specific location within the flash file system, and with a specific name. Without this ability the microkernel can't be loaded into the system, and therefore this system could not be used for microkernel testing.

PC/104 based systems

PC/104 is a public specification for embedding the familiar PC architecture, starting with the i386, into a small form factor (3.6" x 3.8"). The specification details the location and type of connectors used so that components can be combined or inter changed. The advantage of the PC/104 platforms that were surveyed was that many were rugged and geared to industrial applications. However, most systems seemed to include a lot of unnecessary peripherals and with the exception of buying PC/104 data acquisition modules, provided little easy interfacing to IO customization. The PC/104 platforms considered were provided by Arcon, WinSystems, and Diamond Point International Electronics.

Rabbit Semiconductor

Rabbit makes a number of nice development kits, in a variety of ranges and type. There are some very nice platforms with IO and Ethernet. However Rabbit uses its own proprietary microprocessor. While they provide a C compiler, Rabbit systems are not supported by a GNU tool chain.

Gumstix

Gumstix provides a range of embeddable computers powered by an ARM XScale processor. The boards are low in power consumption, but provide good performance. They are small, 80mm x 20mm, and are very affordable. The gumstix boards expose about 80 TTL GPIO lines that can be used to interface to custom devices, such as analog and digital IO for the RTU, and also include on board connectors that allow gumstix accessories to be easily attached. Gumstix accessories include an Ethernet controller, serial UARTs, compact flash and WiFi. Gumstix uses the Buildroot environment, and

the on-board Uboot boot loader can load from built in flash, compact flash cards, a serial connection, or Ethernet connection. The boot loaders command interface is exposed, and the boot loader can even be recompiled and reloaded. The XScale PXA 255 processor is supported by the GNU tool chain, which is used by the build root environment.

7.3 Microkernel Based Hardened RTU Platform

Because of the advantages described above, the Gumstix platform was selected for microkernel evaluation. The Gumstix architecture is relatively affordable and includes native access to TTL GPIO lines that make it possible to add the additional components that will be required in the future for a fully functional RTU prototype. Though a fully functional mircrokernel based RTU is not developed at this point, it is planned that future work will progress in that direction. Gumstix also supports OEM and has been used for a number of commercial devices [122] providing a path for commercialization. Of the many Gumstix boards, the connex 400 was chosen. It is a middle of the road Gumstix board, and is competitively priced at $129. The connex 400 is an XScale PXA 255 processor running at 400 MHz. The XScale processor is a 32 bit processor and the connex 400 has 64 MB ram and 16MB of flash. In addition to the Gumstix motherboard, a Netstix and console ST boards were also purchased. Both of these boards were connected to the connex 400. The Netstix provides an Ethernet controller, which can be used by the boot loader to load images via tftp. The console ST board provides two UART serial interfaces, one of which serves as a console interface to the board.

For the microkernel, the OKL4 implementation was selected as the L4 implementation to use in the microkernel based RTU evaluation. OKL4 is the most

advanced of the implementations, and the implementation is targeted at both researchers and commercial activities. OKL4 is also geared towards embedded systems and RTUs can be considered a type of embedded system. OKL4 supports the XScale processor and the Gumstix platform. In addition to OKL4's L4 implementation, OKL4 includes Iguana. Iguana was developed by NICTA along with their L4 implementation Pistachio-embedded. Iguana is not a complete operating system either, and is intended to work with the underlying L4 implementation to support the development of different L4 based operating systems. Iguana complements, rather than hides, the underlying L4 kernel. Iguana includes an IDL compiler that allows the definition of servers, which provide an interface for Iguana "applications." Access of servers is achieved through L4 IPC, with the IDL just marshalling and un-marshalling parameters.

A Linux 2.6 kernel PC was used as a development platform. The Gumstix development unit was connected to the host development platform using a serial connection and the terminal emulator Kermit. The normal boot processes was interrupted when the Gumstix was powered on, and reconfigured to use tftp to retrieve images from the development host. The OKL4 source code was downloaded and extracted. OKL4 uses a precompiled toolchain, available from NICTA. This tool chain was obtained from [123], and installed on the development system. A test image was then built using the command

```
Tools/build.py machine=gumstix project=iguana test_libs=all
```

The resulting elf image, `image.boot`, was loaded in the gumstix's memory and executed using the `bootelf` command. All tests completed successfully, indicating that the microkernel and iguana were operating correctly on the platform. The next section

describes the development of some RTU software components and IPC performance evaluation of those software components.

7.3 Points Server Development and IPC performance.

With the test platform constructed, the next part of the evaluation was to begin developing RTU components for the test platform. The overall goal of using the L4 microkernel in a hardened RTU is to allow RTU components to be isolated and enable a protection architecture to determine access control. Core RTU components will therefore be implemented as "servers", running as user processes (not in privileged mode). RTU user applications will then call on these services using L4 IPC which passes through a security layer. Figure 7.2 provides a high level view of this architecture.

This architecture creates a hardened platform in several ways. Only the microkernel runs in privileges mode, so the TCB of the RTU is small. RTU services are protected by a non-bypassable security layer. Finally, leveraging the microkernel flexibility, the network drivers and protocol stack used by the application, which in a monolithic kernel must run in the kernel, are mapped into the application address space, where they can at most damage the application instead of the entire kernel.

The L4 IPC provides the path along which the RTU component servers, the security layer and actual RTU programs exchange information and cooperate. A key factor in the RTU's performance will then be the IPC overhead. In particular, the IPC performance of a call from an RTU application to a RTU server. As was discussed in chapter two, IPC is a central component of microkernel based systems that has in the past led to poor system performance. As a central component of an L4 hardened RTU, IPC overhead needs to be low in order to guarantee acceptable RTU performance. An initial

evaluation of IPC overhead was carried out by implementing one of the servers shown in figure 7.2, a limited RTU security layer, and a simple test application program.

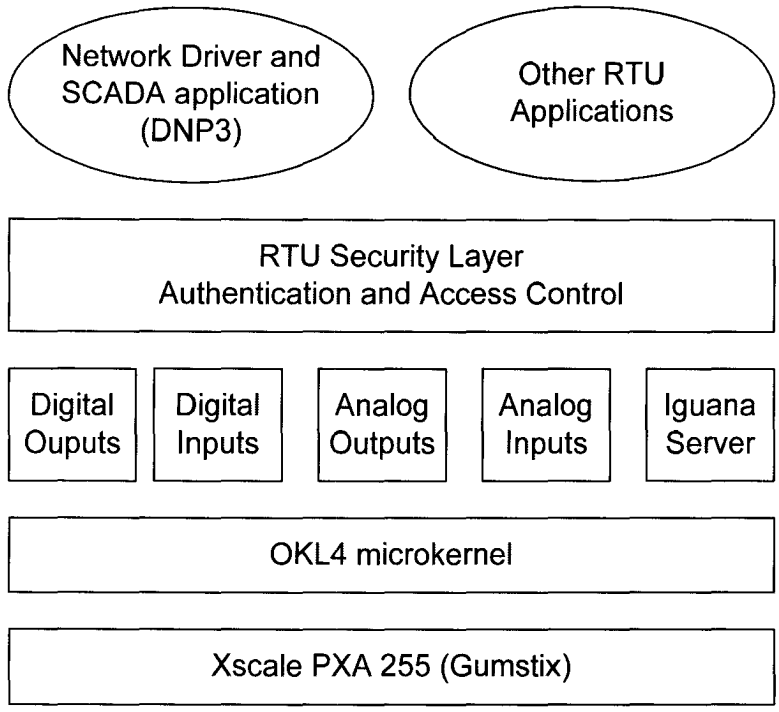


Figure 7.2. Derived OKL4 based RTU architecture.

7.3.1 IPC overhead test setup

To evaluate the IPC performance a server was implemented using Iguana's IDL. The server was called `points`, since it was intended to represent one of the analog or digital IO servers. The server provided an interface to read and write a single analog input, analog input one. Obtaining actual values was not done at this time; instead the analog input value was just stored as a persistent variable. Eventually IO threads that are apart of the analog or digital IO server will update input and output values. The interest at this point was just to measure the IPC overhead.

A security layer and test application were written as an iguana program. The security layer has access to all the underlying RTU servers. The security layer creates the

address space for the RTU applications, and maps into them any needed resources. RTU applications threads are then started and the security then layer listens for IPC requests from the RTU application thread for the low level services provided by the servers.

In the example case, it listens for a request to read analog input one. If the request is allowed, then the operation is performed and the result passed back to the user level thread that made the call. From an IPC perspective this entails several IPC operations as shown in figure 7.3. Initially the RTU security layer thread and the RTU service call IPC_Wait, (1) and (2), which is a blocking IPC that waits for an incoming IPC message. IPC activity is initiated by the application thread's IPC_Send to the RTU security layer thread (3). When the send succeeds, the RTU application thread calls IPC_Receive to wait for the response IPC (4). The RTU security layer then issues an IPC_send to the appropriate RTU server thread (5). The RTU Security layer then has to call IPC_Receive to wait for a response from the server (6). The server responds with an IPC_Send back to the RTU Security layer (7). Finally the RTU security layer calls IPC_Send to the send the response back to the RTU application thread that initiated the IPC sequence (8).

A code fragment from the test application is shown in figure 7.4. A test loop iterates 300 times. Each loop iteration, records the start time and finish time of a loop instance using the `timer_current_time()` call, which is available in Iguana. Iguana's time tick is one microsecond, and the `timer_current_time()` returns the current tick count. Subtracting the final time from the start time gives the number of microseconds that elapsed between (3) and (7). To make sure that IPC operations were indeed reaching the point server and being correctly returned, different values were written to and read from the point server. Observing the results of these read and writes

verified that the RTU application thread was indeed retrieving values from the point server.

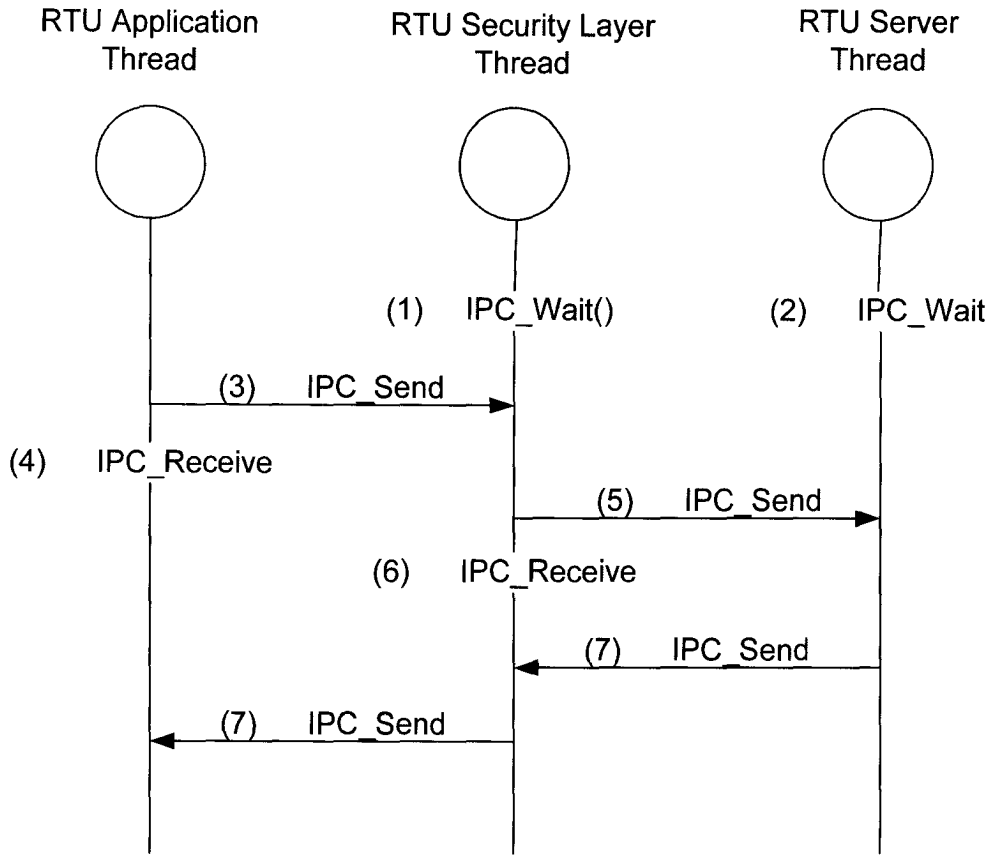


Figure 7.3. IPC calls in an RTU application request for RTU services.

```
#define READ_ANALOG_INPUT_1 0x01
for (i = 0; i < 300; i++)
{
stime = timer_current_time();
L4_MsgClear(&msg);
L4_Set_MsgLabel(&msg,READ_ANALOG_INPUT_1);
L4_MsgLoad(&msg);
tag = L4_Send(thread_l4tid(listener));
assert(L4_IpcSucceeded(tag));
L4_MsgClear(&msg);
tag = L4_Receive(thread_l4tid(listener));
ftime = timer_current_time();
val = L4_Label(tag);
printf("RTU test app read_analog_input_1 call took  %" PRIu64 "
milliseconds, or %" PRIu64
" microseconds\n",((ftime - stime)/1000ULL), (ftime - stime) );
}
```

Figure 7.4. RTU test application code fragment.

7.3.2 IPC overhead results

The elapsed time reported by the code fragment in figure 7.4 also includes the overhead of the `timer_current_time()` call. A separate test program was used where the intermittent L4_IPC calls were removed, leaving just the two calls to `timer_current_time()`. This was used to obtain a measure of the timer overhead, which was determined to be 59.63 microsecond. This was rounded down to 59 microseconds when calculating the actual IPC overhead, so that any error is kept in the IPC overhead.

The above program fragment was executed when the test application was loaded and run on the RTU development test platform. The print statements generate console output, which is output to the development systems screen by Kermit. The test program was run a total of four times. The first value reported each time was several milliseconds, but the remaining sample times were closely grouped around 123 microseconds. This first recorded elapsed time was high because the kernel some one time initialization to do for each thread IPC. Once this is done, it does not have to be done again. Therefore the first recorded time interval was excluded from further calculations. From the remaining times, a total of 500 samples were selected. The mean value was 123.19 microseconds with a standard deviation of .78 and a 95% confidence interval of .002.

Recall that this value still includes the 59 microseconds of the calls to timer overhead. After subtracting this out of the previous result, the actual IPC overhead is for the timer overhead the actual IPC overhead was 64.19 microseconds, for the entire sequence depicted in figure 7.3. Assuming that overhead is evenly distributed, then a single IPC call, which would include a send-to and a receive-from, has an overhead of

approximately 32 microseconds, with a single IPC operation taking about 15 microseconds. These times are indeed much better than the 100 microseconds reported for MACH IPC calls.

Table 7.1. IPC overhead for hardened RTU protected calls.

Description	Value
Average Reported elapsed time	123.19 microseconds
Standard Deviation	.784908
95% confidence interval	.002
Timer overhead	59 microseconds
Actual Average overhead of IPC overhead for hardened RTU protected operation call	64.19 microseconds

7.4 Conclusions

This chapter has presented the results of further investigation of microkernels and their use in developing a hardened RTU. The L4 kernel was selected as a suitable microkernel for RTU development. The address space and thread abstractions provided by L4 make possible the functional and process separation that were identified in chapter five as key objectives in the MILS separation kernel approach. Another advantage of L4 is that it supports real time scheduling. There are several possible L4 implementations. OKL4 was identified as the best L4 implementation for hardened RTU development since it is open source, and supporting both research endeavors and commercial endeavors. The gumstix XScale PXA 255 platform was identified as a good candidate for L4 based hardened RTU development. Gumstix supports OEM activities and is already in use in some commercial activities. Gumstix's architecture is open and accessible, and is supported by an open source build system including GNU tool chain.

The microkernel based RTU will include a security layer that protects low-level RTU services implemented as L4 or Iguana servers. Hardened RTU application will use

L4 IPC to access these servers, through IPC to the security layer. Some initial development on an ARM XScale platform, loaded with the OKL4 kernel, was done to determine what the IPC overhead would be for the hardened RTU. Platform evaluation found the IPC overhead to be 64.19 microseconds.

CHAPTER VIII

CONCLUSIONS AND FUTURE DIRECTIONS

SCADA and DCS are used in a wide variety of utility and industrial operations. Complete isolation of these control networks is no longer a possible or plausible means of providing cyber security for these systems. Instead, security must be built into SCADA systems. However, control systems differ from traditional IT systems in a number of ways. One of those differences is the importance of securing perimeter devices, such as RTUs, in control networks. This dissertation has developed a new role based access control model for RTUs, and investigated two approaches for using minimal kernels, all of which can be used to create a security hardened RTU. This chapter presents the overall conclusions from this dissertation research and possible directions for future research.

8.1 Conclusions

A major focus of current SCADA security efforts has been authentication. This research has argued that RTUs need to provide access control as well. The insider threat and increased commercial RTU software components, such as web servers, are two important threats that RTU access control can help reduce. This dissertation has presented a new role based access control model designed specifically for RTUs and process control. The model includes assignment constraints based on RTU operation type which limit the RTU operations that a specific role can be assigned. The model also

includes activation constraints that allow a security administrator to specify conditions when specific roles or permissions cannot be used. Constraints are activated by context information, including process control specific state information as well as the process control relevant information of location and time.

The RTU access control model helps to mitigate the insider threat by allowing those who must have access to the RTU, to be limited to only those RTU operations that their duties require. Contractors and business partners, or new employees can easily be given reduced privileges. Organizing RTU permissions around roles also makes it much easier to manage the assignment and revocation of permissions to users, and decrease the probability of a misconfiguration or forgotten privilege. The assignment constraints, based on RTU operation type of *control*, *status*, and *configuration* allow a security administrator to establish separation of duty policies that the model then makes sure are not violated, further reducing the chance of a misconfiguration or inappropriate assignment of permissions. Context based constraints, such as "Bill cannot activate his engineering role on weekends, or at night", allow fined grained control of permissions organized around logical contexts.

Additional RTU hardening can be achieved through development of a minimal kernel RTU and this dissertation has investigated two possible approaches. The first approach is to reduce a commercial OS kernel to only those components needed by the RTU, eliminating known and unknown vulnerabilities and significantly reducing the size of the kernel. The second approach proposes using a microkernel that supports partitioning as the basis for an RTU specific operating system which isolates network related RTU software, from critical RTU operation software such as local control and

analog and digital input and output. This isolation provides an additional, and nearly impenetrable layer of separation between software that can be attacked and critical RTU software. Allowed channels of communication are achieved and enforced using kernel IPC. The use of IPC allows security critical RTU code to be protected, and makes circumvention of security less likely.

A prototype hardened RTU was constructed using the reduced COTS kernel approach and implementing the newly developed RTU role based access control model. The over 50% reduction in the size of the commercial kernel confirms that significant amounts of code in commercial kernels can be eliminated. This eliminated code can contain both known and unknown vulnerabilities, increasing the RTU's security. The prototype was connected to real SCADA hardware in the Chemical Engineering Department's Process Control Laboratory. Functional testing of users, roles and constraints confirmed that users were able to carryout assigned tasks with the limited set of permissions provided by the security hardened RTU. Security tests, ranging from scanning and network based attacks to simulated insider stacks were all positive, demonstrating the RTUs increased resistance to cyber based attacks. Analysis of the communication times found the prototype RTU response time to be within 500 milliseconds, acceptable for many SCADA and DCS application areas. This indicates that the RTU access controls and reduced kernel did not negatively impact performance. Potential commercialization of this prototype is possible after additional testing and refinement.

Investigation of a partitioning microkernel for an RTU identified the L4 microkernel as a potential candidate since L4 provides partitioning and real time

scheduling. To evaluate L4, the OKL4 implementation was used on an embedded ARM XScale processor. Using a Gumstix development platform, the microkernel approach was demonstrated by creating simulated critical RTU operations that were isolated from network facing software using L4 address spaces. The approach makes extensive use of IPC and could be negatively impacted by high IPC overhead. Experimental analysis found the IPC overhead for protected RTUs operations to be slightly less than 65 microseconds, sufficiently small to warrant continued investigation of an OKL4/ARM based hardened RTU.

8.2 Future Research Directions

The RTU role based access control model chose to use negative constraints. The decision was made to strike a balance between allowing, through role assignment, and denying, through constraints. A different approach, one that focuses entirely on instantaneous enabling of permissions could be explored. In such an approach, RTU permission assignment would be completely event driven. Such an approach would result in an even better approximation of least privilege, allowing only those permissions needed at any moment to be granted to a subject. Just as important, the approach makes sure that the access control is continuous, allowing an attack to be prevented even if initial condition under which the attack occurred would have allowed it to succeed.

A second direction of research related to the RTU access control policy is to translate the policy into constraint logic programming (CLP). The use of CLP is gaining interest in access control research due to the ability to apply AI reasoning research to the domain. A CLP based model should be equally expressive, and could encourage the development of policy tools. The development of tools to help SCADA operators

configure and reason about policies would be of great value. SCADA systems are complex systems, and as they grow in size and intricacy, it will be very difficult to develop and analyze policies. Development of tools for policy development and policy checking, especially CLP, is an area of exploration that might yield results that could directly benefit the SCADA community and possibly be translated to other commercial sectors.

The ability to express constraints is one major advantage of role based access control models over the traditional access control matrix model. Another strength of role based access control that could be exploited by this model is role hierarchies. In role hierarchies, more privileges roles inherit the permissions of more junior roles, forming a lattice of roles. This allows roles to be layered and permissions to be grouped not only by role but by their dominance or subordination to other permissions. The application of role hierarchies to the RTU role based access control model could provide even more logical permission grouping and is worthy of further investigation, though it might entail the addition of more roles to the model.

Another direction for future research is related to handling access denials. The current prototype returns a flag to the application that the access was denied. The application then has to handle the error. In DNP3 this could be done using the internal indicator status bytes to indicate the point is offline. A more elegant and potentially much more secure approach is to explore having the RTU provide something like poly-instantiation of RTU points and services. By applying a type of poly-instantiation, a connected user or device would only see those points (eventually other RTU services as well) to which they had access. It would be as if users with different roles or

authorization were connected to different RTUs. This would discourage attackers by making less privilege scans of the system show little of interest. Poly-instantiation could also assist in assuring availability by helping clearly define how the RTU should be partitioned to maximize the ease of implementing the RTU poly-instantiation.

The potential security benefits of microkernel based systems are just now beginning to be seriously considered. The key challenge in furthering this agenda is developing systems that provide specific security architectures. While an L4 based RTU has a smaller TCB, and can isolate components, the need for component interaction mandates more elaborate security support. For the hardened RTU based microkernel, the next area of exploration is to demonstrate that the RTU role based access control model can be implemented on L4, and that it can be done so with in a more secure way than on a COTS system. A static policy can easily be implemented by the security middleware layer, but what will eventually be desirable is for objects to protect themselves using the security server. This kind of protection allows program developers to include security during the entire software development lifecycle. This is a goal for most software development activities, but is doubly critical for RTUs and other field devices, and the potential consequences of security violations can be dire.

The harsh environment in which RTUs operate and the importance of availability provide motivation for a final future research direction. The measures presented in this dissertation focus on protection, and did not include any performance monitoring features. However, the incorporation of performance monitoring into the security harden RTU would enhance security, especially availability. With the performance monitoring, the RTU may be able to self report problems before they become big problems.

Performance features could give additional indication of a security violation, or may just indicate a physical problem. Possible performance monitoring features would be processor temperature, the number of packet reassembly failures, the longest and average time for a request to be received, the number of processes currently active and/or inactive. Performance monitoring comes at a cost, both for the extra cycles needed to calculate and store performance data, and the bandwidth needed to transmit the extra data. Further investigation is needed to determine the most appropriate performance measures, and how to assure that they do not themselves create vulnerabilities.

REFERENCES

[1] D. Geer, "Security of Critical Control Systems Sparks Concern," *Computer*, vol. 39, no. 1, pp. 20-23, 2006.

[2] "Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism (USA PATRIOT ACT) Act of 2001,", H.R. 3162 ed 2001.

[3] President's Commission on Critical Infrastructure Protection, "Critical Foundations: Protecting America's Infrastructures," United States Government Printing Office (GPO),No. 040-000-00699-1, Oct.1997.

[4] Poulsen, Kevin, "Slammer worm crashed Ohio nuke plant net", The Register. http://www.theregister.co.uk/2003/08/20/slammer_worm_crashed_ohio_nuke/. (accessed on 5-6-2006).

[5] Smith, Tony, "Hacker Jailed for Revenge sewage attacks", http://www.theregister.co.uk/2001/10/31/hacker_jailed_for_revenge_sewage/. (accessed on 3-15-2006).

[6] E. Byres and J. Lowe, "The Myths and Facts behind Cyber Security Risks for Industrial Control Systems," *VDE Kongress, Berlin, Germany*, 2004.

[7] IEEE Std.C37.1-1994, "IEEE Standard Definition, Specification, and Analysis of Systems Used for Supervisory Control, Data Acquisition, and Automatic Control," 1994.

[8] "SCADA RTU's", http://members.iinet.net.au/~ianw/rtu.html. (accessed on 12-15-2007).

[9] R. Carlson, "Sandia SCADA Program High-Security SCADA LDRD Final Report," Sandia National Labs, Alburquerque, New Mexico, SANDIA Technical Report SAND2002-0729, 2002.

[10] D. J. Gaushell and H. T. Darlington, "Supervisory control and data acquisition," *Proceedings of the IEEE*, vol. 75, no. 12, pp. 1645-1658, 1987.

[11] W. J. Ackerman and W. R. Block, "Understanding supervisory systems," *Computer Applications in Power, IEEE*, vol. 5, no. 4, pp. 37-40, 1992.

[40] Peterson, D., "Important New Nessus Plugin for ICCP Users", http://www.digitalbond.com/index.php/2007/03/01/important-nessus-plugin-for-iccp-users/.

[41] P. Oman, E. Schweitzer, and D. Frincke, "Concerns About Intrusions into Remotely Accessible Substation Controllers and SCADA Systems," *27th Annual Western Protective Relay Conference, Paper*, vol. 4, pp. 23-26.

[42] D. Dolezilek, K. Carson, K. Leech, K. Streett, and O. P. D. No, "SECURE SCADA AND ENGINEERING ACCESS COMMUNICATIONS: A CASE STUDY OF PRIVATE AND PUBLIC COMMUNICATION LINK SECURITY,".

[43] E. J. Byres, "Designing Secure Networks for Process Control," *Conference Record of 1999 Annual Pulp and Paper Industry Technical Conference*, pp. 63-67, June1999.

[44] J. Abshier, "10 Principles for securing control systems," *Control*, vol. 18, no. 10, pp. 77-81, 2005.

[45] J. D. Fernandez and A. E. Fernandez, "SCADA systems: vulnerabilities and remediation," *Journal of Computing Sciences in Colleges*, vol. 20, no. 4, pp. 160-168, 2005.

[46] K. Stouffer, J. Falco, and K. Kent, "Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security," NIST,Special Publication 800-82 INITIAL PUBLIC DRAFT, Sept.2006.

[47] "Process Control Security Requirements Forum (PCSRF)", http://www.isd.mel.nist.gov/projects/processcontrol/.

[48] "Field Device Protection Profile For SCADA Systems In Meduim Robust Environments", http://www.isd.mel.nist.gov/projects/processcontrol/FieldDevicePP/Field_Device_PP_0_71.pdf.

[49] M. Majdalawieh, F. Parisi-Presicce, and D. Wijesekera, "DNPsec: A Secure Framework for DNP3 in SCADA systems," in *International Joint Conference on Computer Information and Systems Sciences and Engineering* 2005.

[50] "Cryptographic Protection of SCADA Communications Part 1: Background, Policies and Test Plan", http://www.gtiservices.org/security/AGA%2012%20Part%201%20Final%20Version.pdf.

[51] "Concept of Operations," American Gas Association,AGA 12-2, Mar.2006.

[52] A. K. Wright, J. A. Kinast, and J. McCarty, "Low-Latency Cryptographic Protection for SCADA Communications," in *Applied Cryptography and Network Security,* 3089 ed Springer Berlin, 2004, pp. 263-277.

[53] C. L. Beaver, D. R. Gallup, W. D. NeuMann, and M. D. Torgerson, "Key Management for SCADA," *Cryptog. Information Sys. Security Dept. , Sandia Nat. Labs, Tech. Rep. SAND2001-3252, Mar*, 2002.

[54] P. Palensky and T. Sauter, "Security considerations for FAN-Internet connections," *Factory Communication Systems, 2000. Proceedings. 2000 IEEE International Workshop on*, pp. 27-35, 2000.

[55] C. L. Bowen III, T. K. Buennemeyer, and R. W. Thomas, "Next generation SCADA security: best practices and client puzzles," *Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005. Proceedings from the Sixth Annual IEEE*, pp. 426-427, 2005.

[56] M. Naedele and O. Biderbost, "Human-assisted intrusion detection for process control systems," *Proc. 2nd Int. Conf. Applied Cryptography and Network Security*, pp. 216-225, 2004.

[57] E. Naess, D. A. Frincke, A. D. McKinnon, and D. E. Bakken, "Configurable Middleware-Level Intrusion Detection for Embedded Systems," *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pp. 144-151, 2005.

[58] E. J. Byres, M. Franz, and D. Miller, "The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems," *International Infrastructure Survivability Workshop (IISW'04), IEEE, Lisbon, Portugal, December*, vol. 4 2004.

[59] D. Conte de Leon, J. Alves-Foss, A. Krings, and P. Oman, "Modeling Complex Control Systems to Identify Remotely Accessible Devices Vulnerable to Cyber Attack," *ACM Workshop on Scientific Aspects of Cyber Terrorism,(SACT)*, 2002.

[60] W. F. Young, J. E. Stamp, J. D. Dillinger, and M. A. Rumsey, "Communication Vulnerabilities And Mitigations In Wind Power SCADA Systems," in *American Wind Energy Association WINDPOWER 2003 Conference* 2003.

[61] M. Franz and V. Pothamsetty, "ModbusFW Deep Packet inspection for industrial Ethernet,".

[62] Z. Xie, G. Manimaran, V. Vittal, A. G. Phadke, and V. Centeno, "An information architecture for future power systems and its reliability analysis," *Power Systems, IEEE Transactions on*, vol. 17, no. 3, pp. 857-863, 2002.

[63] C. H. Hauser, D. E. Bakken, and A. Bose, "A failure to communicate: next generation communication requirements, technologies, and architecture for the

electric power grid," *Power and Energy Magazine, IEEE*, vol. 3, no. 2, pp. 47-55, 2005.

[64] P. J. DENNING, "Third Generation Computer Systems," *Computing Surveys*, vol. 3, no. 4, pp. 175-216, 1971.

[65] D. E. BELL and L. J. LAPADULA, "Secure computer system: Unified exposition and multics interpretation," Mitre Corportation, Bedford MA,MTR-2997, 1976.

[66] E. Bertino, "RBAC models- concepts and trends," *Computers & Security*, vol. 22, no. 6, pp. 511-514, 2003.

[67] D. F. FERRAIOLO, R. SANDHU, S. GAVRILA, D. R. KUHN, and R. CHANDRAMOULI, "Proposed NIST Standard for Role-Based Access Control," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 224-274, 2001.

[68] R. Simon and M. E. Zurko, "Separation of duty in role-based environments," *Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, 1997.

[69] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Communincation of the ACM*, vol. 17, no. 7 1975.

[70] D. Clark and David Wilson, "Comparison of commercial and military computer security," 1987, pp. 184-194.

[71] G. J. Ahn, "Specification and classification of role-based authorization policies," 2003, pp. 202-207.

[72] G. J. Ahn and R. SANDHU, "Role-based authorization constraints specification," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 207-226, 2000.

[73] M. Strembeck and G. Neumann, "An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments," *ACM Transactions on Information and System Security*, vol. 7, no. 3, pp. 392-427, 2004.

[74] T. Jaeger, "On the increasing importance of constraints," *Proceedings of the fourth ACM workshop on Role-based access control*, pp. 33-42, 1999.

[75] L. Giuri and P. Iglio, "A formal model for role-based access control with constraints," 1996, pp. 136-145.

[76] L. Giuri and P. Iglio, "A New Model for Role Based Access Control," 1995.

[77] M. A. Bishop, *Computer Security: Art and Science*. New York: Addison-Wesley, 2003.

[78] N. Li, Z. Bizri, and M. V. Tripunitara, "On mutually-exclusive roles and separation of duty," *Proceedings of the 11th ACM conference on Computer and communications security*, pp. 42-51, 2004.

[79] R. SANDHU, V. Bhamidipati, and Q. MUNAWER, "The ARBAC97 model for role-based administration of roles," *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 1, pp. 105-135, 1999.

[80] F. Chen and R. S. Sandhu, "Constraints for role-based access control," *Proceedings of the first ACM Workshop on Role-based access control*, 1996.

[81] S. Wook, J. Y. Park, and L. E. E. Dong-Ik, "Extended Role Based Access Control with Procedural Constraints for Trusted Operating Systems," *IEICE Transactions on Information and Systems*.

[82] J. Crampton, "Specifying and enforcing constraints in role-based access control," *Proceedings of the eighth ACM symposium on Access control models and technologies*, pp. 43-50, 2003.

[83] G. Neumann and M. Strembeck, "An Approach to Engineer and Enforce Context Constraints in an RBAC Environment," *Proceedings of the eighth ACM symposium on Access control models and technologies*, pp. 65-79, 2003.

[84] A. S. Tanenbaum, J. N. Herder, and H. Bos, "Can we make operating systems reliable," *Computer*, vol. 39, no. 5, pp. 44-51, May2006.

[85] G. Heiser, K. Elphinstone, I. Kuz, G. Klein, and S. M. Petters, "Towards Trustworthy Computing Systems: Taking Microkernels to the Next Level," *Operating Systems Review*, vol. 41, no. 3 2007.

[86] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell, "The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments," *Proceedings of the 21st National Information Systems Security Conference*, vol. 314 1998.

[87] J. Liedtke, "On micro-kernel construction," *ACM SIGOPS Operating Systems Review*, vol. 29, no. 5, pp. 237-250, 1995.

[88] P. B. HANSEN, "The Nucleus of a Multiprogramming System," *Operating Systems*, 1970.

[89] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "HYDRA: The Kernel of a Multiprocessor Operating System," *Communications*, 1974.

[90] R. Rashid, A. Tevanian, Jr., M. Young, D. Golub, R. Baron, D. Black, W. J. Bolosky, and J. Chew, "Machine-independent virtual memory management for

paged uniprocessor and multiprocessor architectures," *Transactions on Computers*, vol. 37, no. 8, pp. 896-908, 1988.

[91] "http://en.wikipedia.org/wiki/NEXTSTEP," 2007.

[92] "http://www.cs.utah.edu/flux/mach4/html/Mach4-proj.html," 2007.

[93] "http://en.wikipedia.org/wiki/Workplace_OS," 2007.

[94] J. B. Chen and B. N. Bershad, "The impact of operating system structure on memory system performance," *Proceedings of the fourteenth ACM symposium on Operating systems principles*, pp. 120-133, 1994.

[95] J. Liedtke, "Improving IPC by kernel design," *Proceedings of the fourteenth ACM symposium on Operating systems principles*, pp. 175-188, 1994.

[96] J. M. Rushby, "Design and verification of secure systems," *Proceedings of the eighth ACM symposium on Operating systems principles*, pp. 12-21, 1981.

[97] W. M. Vanfleet, J. A. Luke, R. W. Beckwith, C. Taylor, B. Calloni, and G. Uchenick, "MILS:Architecture for High-Assurance Embedded Computing," *CrossTalk The Journal of Defense Software Engieering*, Aug.5 A.D.

[98] J. Rushby, "Kernels for safety," *Safe and Secure Computing Systems*, pp. 210-220, 1989.

[99] N. Hanebutte, P. Oman, M. Loosbrock, A. Holland, W. S. Harrison, and J. ves-Foss, "Software mediators for transparent channel control in unbounded environments," *Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005. Proceedings from the Sixth Annual IEEE*, pp. 201-206, 2005.

[100] Uchenick, Gordon, "MILS: Architecture for High-Assurance Systems", Objective Interface Systems. http://www.rtcmagazine.com/home/printthis.php?id=100319.

[101] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "An access control model supporting periodicity constraints and temporal reasoning," *ACM Transactions on Database Systems (TODS)*, vol. 23, no. 3, pp. 231-285, 1998.

[102] "Cert Advisory CA 2001-33", http://www.cert.org/advisories/CA-2001-33.html. http://www.cert.org/advisories/CA-2001-33.html.

[103] A. S. Tanenbaum, J. N. Herder, and H. Bos, "Can we make operating systems reilable," *Computer*, vol. 39, no. 5, pp. 44-51, May2006.

[104] "The Keyed-Hash Message Authentication Code (HMAC)," NIST,FIPS PUB 198, Mar.2002.

[105] J. L. Hieb, S. C. Patel, and J. H. Graham, "Security Enhancements for Distributed Control Systems," in *Critical Infrastructure Protection: Issues and Soutions*. S. Shenoi and E. Goetz, Eds. Boston: Springer, 2007.

[106] M. A. Bishop, *Computer Security: Art and Science* Addison-Wesley Professional, 2002.

[107] D. E. Denning, "A Lattice Model of Secure Information Flow," *Communications*, 1976.

[108] R. S. Sandhu, "Lattice-based access control models," *Computer*, vol. 26, no. 11, pp. 9-19, 1993.

[109] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38-47, 1996.

[110] V. R. Basili and B. T. Perricone, "Software errors and complexity: an empirical investigation 0," *Communications of the ACM*, vol. 27, no. 1, pp. 42-52, 1984.

[111] "http://www.sixnetio.com/html_files/products_and_groups/mipm_vt.htm," 2007.

[112] "LynuxWorks", http://www.lynuxworks.com/. http://www.lynuxworks.com/.

[113] "http://en.wikipedia.org/wiki/Coordinated_Universal_Time," 2007.

[114] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed Hashing for Message Authentication," Network Working Group,RFC 2104, Feb.1997.

[115] W. F. Rush and A. Shah, "Impact of Information Security Systems on Real-Time Process Control," Gas Technology Institute,NIST Project SB1341-02-C-081, Apr.2005.

[116] J. Liedtke, "Toward real microkernels," *Communications of the ACM*, vol. 39, no. 9, pp. 70-77, 1996.

[117] "http://l4ka.org/projects/pistachio/," 2007.

[118] " http://os.inf.tu-dresden.de/fiasco/," 2007.

[119] "http://www.ok-labs.com/," 2007.

[120] "Open Kernel Labs", http://www.ok-labs.com/. (accessed on 12-15-2007).

[121] "http://www.ok-labs.com/company/press_releases/ok_10_09_07," 2007.

[122] "http://docwiki.gumstix.org/Customer_projects_commercial_2," 2007.

[123] "http://www.ertos.nicta.com.au/downloads/tools/arm-linux-3.4.4.tar.gz," 2007.

GLOSSARY

Certificate Authority (CA) – Entity that creates and issues digital certificates used by other parties in a PKI.

Commercial Off The Self (COTS) – Described ready made products that are easily obtainable. Products include both hardware and software, and the use of open standards is an important component. An excellent example of COTS hardware would be standard Ethernet cards. Examples of COTS software products include operating systems like Windows, Linux, and MAC OS as well as server products such as IIS or Apache, and user application like word processors or compilers. COTS components are typically inexpensive compared to the cost of custom developed products.

Common Object Request Broker Architecture (CORBA) – A standard created and controlled by OMG that defines APIs and communications protocols that allow heterogeneous software components to interoperate.

Denial of Service (DoS) – Inhibition of a service or resource for an extended period of time. A type of attack which usually does not result in the loss of information or corruption of information, but which prevents valid users from accessing a service or resource

Distributed Control System (DCS) – A distributed PCS where one or more sub-controllers at different geographic location within the plant are monitored and controlled from a singe remote location.

Distributed Network Protocol (DNP3) – A SCADA communication protocol for delivering the status of field equipment from RTUs to MTUs and control commands from MTUs to RTUs. The primary abstraction used in the DNP3 protocol is that of points where a point indicated a specific value associated with a specific piece of field equipment. The protocol is broken into two components the client and the server. Typically RTUs implement the DNP3 server protocol and MTUs implement DNP3 client protocol.

Field Equipment – In SCADA and process control systems field equipment refers to devices measure or operate physical system. Temperature sensors and value controllers are excellent examples of field equipment. Field equipment may have a communication interface, in which case it might be referred to as a IED. Other field equipment may have only analog and digital leads.

Fieldbus – The network that links sensors, actuators, and other devices to a PC or PLC based controller eliminating the need for point to point wiring of device to controller.

File Transfer Protocol (FTP) – A network protocol for exchanging files over a network.

Firewall – A combination of hardware and software components that inhibit network traffic flows based on security policy.

Human Machine Interface (HMI) – The hardware or software through which the an operator interacts with a controller. HMIs range in complexity from a physical panel with buttons and lights to an industrial PC with color graphics running HMI software.

Independent Application Development Kit (IADK) – A set of tools and libraries from SIXNET for custom software development for mIMP based RTUs.

Intelligent Electronic Device (IED) – Any device that incorporates one or more processors capable of sending or receiving data or control to or from an external source.

Inter Process Communication (IPC) – A set of techniques for exchanging data between two or more processes running on the same or different computers.

Internet Protocol (IP) – An open network layer protocol used for communication data across packet-switched network.

Internet Protocol Security (IPsec) – A set of protocol developed to support secure exchange of packets at the IP layer. Especially useful in implementing VPNs.

Local Area Network (LAN) – A group of computers that share a common communicatins line and occupy a relatively small geographic area (such as a build or control room). A LAN can consist of a couple of computers to several thousand. Ethernet is by far the most common LAN network technology.

Master Terminal Unit (MTU) – A SCADA component whose primary responsibility is to maintain a real-time data about the status of the system through regular polling of RTUs. MTUs are also responsible for sending operator control signal back to RTUs. Some MTUs provide a user interface and in other situations the user interface is provided by a separate machine usually referred to as a HMI.

Modbus – A open standard SCADA protocol for RTU – MTU communication.

Multiple Independent Layers of Security (MILS) – A high-assurance high-performance computing architecture that can enforce strict security and separation policies on data and processes residing on a single microprocessor.

Partition Communication System (PCS) – Part of the MILS standard that is under development. Partitions are the unit of separation in MILS and the PCS is a middleware layer that enforces a security policy on communication between partitions on one or more processors.

Policy Decision Point (PDP) – An access control entity that receives requests from the policy enforcement point in the form of (subject, object, operations) and returns either true or false indicating whether a given request is allowed by the policy.

Policy Enforcement Point (PEP) – An access control entity that is logically located between subjects and systems resources, and enforces the systems access control policy.

Process Control System (PCS) – A computer system that processes sensor inputs, executes control algorithms, and computes actuator outputs. In a PCS control decisions are made by the computer system based on control algorithms.

Programmable Logic Controller (PLC) – A small industrial computer used in factories originally designed to replace relay logic of a process control system and has evolved into a controller having the functionality of a process controller.

Public Key Infrastructure (PKI) – A cryptographic arrangement that provides for third party validation of user identities by tying public keys to a user identity.

Public Switched Telephone Network (PSTN) – The international telephone system that uses copper wire to transmit analog data, usually voice communication.

Remote Terminal Unit (RTU) – Also known as remote telemetry unit. SCADA system component that acquires data from sensors, delivers control signals to field equipment, and communicates with the master station.

Role Based Access Control (RBAC) – An access control scheme in which the permission to carryout various operations are assigned to roles. A user of the system is then assigned to one or more roles and is allowed to carryout only those operations which are associated with the users roles.

SCADA Cryptographic Module (SCM) – A hardware device that provides secure serial SCADA communication between MTUs and RTUs. The SCM has two ports a plaintext port and a ciphertext port. SCADA messages from either an RTU or MTU are received on the SCM's plaintext port and protected messages are sent out the cipher text port and vice versa. Secure communication requires a SCM at each end of the communications channel.

Secure Shell (SSH) – A protocol that provides for the establishment of a secure channel between a local and remote computer using public-key cryptography and symmetric encryption.

Serial SCADA Protection Protocol (SSPP) – A cryptographic protocol developed by the AGA as part of its effort to secure SCADA systems. The protocol is designed for serial communications and the ability to support many SCADA protocols.

Supervisory Control And Data Acquisition System (SCADA) – Provides the ability to monitor and control the operation of a distributed physical system, such as electrical power distribution, from a single remote location. In general control commands are delivered by the SCADA system but control decision are made by a human operator in response to alarms generated by the SCADA system.

Transmission Control Protocol (TCP) – A transport layer protocol that provides applications with a reliable stream for in order data flow.

Trusted Computing Base (TCB) – The collection of hardware and software components that are involved in enforcing security for a given computing system.

Universal Coordinated Time (UTC) – Greenwich mean time.

Virtual Local Area Network (VLAN) – A network of computers that behave as if they were connected to the same physical network segment of a LAN. VLANs are configured through software.

Virtual Private Network (VPN) – A private communications network used by one or more organizations to communicate over a public network.

War-dialer – a computer program that uses a modem connected to the computer to dial hundreds or thousands of phone numbers and identifies numbers to which a corresponding modem and computer are connected.

Wide Area Network (WAN) – A computer network that spans a large geographical area, such as multiple states or cities. Computer connected to the WAN are often connected through public networks like the telephone system.

APPENDIX A

REDUCED LINUX KERNEL CONFIGURATION FILE

```
#
# Automatically generated by make menuconfig: don't edit
#
# CONFIG_UID16 is not set
# CONFIG_RWSEM_GENERIC_SPINLOCK is not set
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_HAVE_DEC_LOCK=y

#
# Code maturity level options
#
CONFIG_EXPERIMENTAL=y

#
# Loadable module support
#
CONFIG_MODULES=y
# CONFIG_MODVERSIONS is not set
CONFIG_KMOD=y

#
# Platform support
#
CONFIG_PPC=y
CONFIG_PPC32=y
# CONFIG_6xx is not set
# CONFIG_4xx is not set
# CONFIG_POWER3 is not set
# CONFIG_POWER4 is not set
CONFIG_8xx=y
# CONFIG_PPC_STD_MMU is not set
CONFIG_SERIAL_CONSOLE=y
# CONFIG_RPXLITE is not set
# CONFIG_RPXCLASSIC is not set
# CONFIG_BSEIP is not set
# CONFIG_FADS is not set
CONFIG_SXNI855T=y
# CONFIG_TQM823L is not set
# CONFIG_TQM850L is not set
# CONFIG_TQM855L is not set
# CONFIG_TQM860L is not set
# CONFIG_FPS850L is not set
# CONFIG_TQM860 is not set
# CONFIG_SPD823TS is not set
# CONFIG_IVMS8 is not set
# CONFIG_IVML24 is not set
# CONFIG_SM850 is not set
# CONFIG_MBX is not set
# CONFIG_WINCEPT is not set
```

```
# CONFIG_ALL_PPC is not set
# CONFIG_SMP is not set
CONFIG_MATH_EMULATION=y

#
# General setup
#
# CONFIG_HIGHMEM is not set
# CONFIG_ISA is not set
# CONFIG_EISA is not set
# CONFIG_SBUS is not set
# CONFIG_MCA is not set
# CONFIG_PCI_QSPAN is not set
# CONFIG_PCI is not set
CONFIG_HZ=100
CONFIG_NET=y
CONFIG_SYSCTL=y
CONFIG_SYSVIPC=y
# CONFIG_BSD_PROCESS_ACCT is not set
CONFIG_KCORE_ELF=y
CONFIG_BINFMT_ELF=y
CONFIG_KERNEL_ELF=y
# CONFIG_BINFMT_MISC is not set
# CONFIG_HOTPLUG is not set
# CONFIG_PCMCIA is not set

#
# Parallel port support
#
# CONFIG_PARPORT is not set
CONFIG_PPC_RTC=y
CONFIG_CMDLINE_BOOL=y
CONFIG_CMDLINE="console=ttyS0,9600 console=tty0 root=/dev/sda2"

#
# Memory Technology Devices (MTD)
#
CONFIG_MTD=y
# CONFIG_MTD_DEBUG is not set
CONFIG_MTD_PARTITIONS=y
# CONFIG_MTD_CONCAT is not set
# CONFIG_MTD_REDBOOT_PARTS is not set
# CONFIG_MTD_CMDLINE_PARTS is not set
CONFIG_MTD_CHAR=y
CONFIG_MTD_BLOCK=y
# CONFIG_FTL is not set
# CONFIG_NFTL is not set

#
# RAM/ROM/Flash chip drivers
#
CONFIG_MTD_CFI=y
CONFIG_MTD_JEDECPROBE=y
CONFIG_MTD_GEN_PROBE=y
# CONFIG_MTD_CFI_ADV_OPTIONS is not set
CONFIG_MTD_CFI_INTELEXT=y
CONFIG_MTD_CFI_AMDSTD=y
```

```
# CONFIG_MTD_CFI_STAA is not set
# CONFIG_MTD_RAM is not set
# CONFIG_MTD_ROM is not set
# CONFIG_MTD_ABSENT is not set
# CONFIG_MTD_OBSOLETE_CHIPS is not set
# CONFIG_MTD_AMDSTD is not set
# CONFIG_MTD_SHARP is not set
# CONFIG_MTD_JEDEC is not set

#
# Mapping drivers for chip access
#
# CONFIG_MTD_PHYSMAP is not set
CONFIG_MTD_SXNI855T=y
# CONFIG_MTD_TQM8XXL is not set
# CONFIG_MTD_RPXLITE is not set
# CONFIG_MTD_MBX860 is not set
# CONFIG_MTD_DBOX2 is not set
# CONFIG_MTD_CFI_FLAGADM is not set
# CONFIG_MTD_REDWOOD is not set
# CONFIG_MTD_PCI is not set
# CONFIG_MTD_PCMCIA is not set

#
# Self-contained MTD device drivers
#
# CONFIG_MTD_PMC551 is not set
CONFIG_MTD_SLRAM=y
# CONFIG_MTD_MTDRAM is not set
# CONFIG_MTD_BLKMTD is not set
# CONFIG_MTD_DOC1000 is not set
# CONFIG_MTD_DOC2000 is not set
# CONFIG_MTD_DOC2001 is not set
# CONFIG_MTD_DOCPROBE is not set

#
# NAND Flash Device Drivers
#
CONFIG_MTD_NAND=y
CONFIG_MTD_NAND_VERIFY_WRITE=y
CONFIG_MTD_NAND_SXNI=y
CONFIG_MTD_NAND_IDS=y

#
# Plug and Play configuration
#
# CONFIG_PNP is not set
# CONFIG_ISAPNP is not set

#
# Block devices
#
# CONFIG_BLK_DEV_FD is not set
# CONFIG_BLK_DEV_XD is not set
# CONFIG_PARIDE is not set
# CONFIG_BLK_CPQ_DA is not set
# CONFIG_BLK_CPQ_CISS_DA is not set
```

```
# CONFIG_BLK_DEV_DAC960 is not set
# CONFIG_BLK_DEV_LOOP is not set
# CONFIG_BLK_DEV_NBD is not set
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_SIZE=4096
CONFIG_BLK_DEV_INITRD=y

#
# Multi-device support (RAID and LVM)
#
# CONFIG_MD is not set
# CONFIG_BLK_DEV_MD is not set
# CONFIG_MD_LINEAR is not set
# CONFIG_MD_RAID0 is not set
# CONFIG_MD_RAID1 is not set
# CONFIG_MD_RAID5 is not set
# CONFIG_MD_MULTIPATH is not set
# CONFIG_BLK_DEV_LVM is not set

#
# Networking options
#
# CONFIG_PACKET is not set
# CONFIG_NETLINK_DEV is not set
# CONFIG_NETFILTER is not set
# CONFIG_FILTER is not set
CONFIG_UNIX=y
CONFIG_INET=y
# CONFIG_IP_MULTICAST is not set
# CONFIG_IP_ADVANCED_ROUTER is not set
# CONFIG_IP_PNP is not set
# CONFIG_NET_IPIP is not set
# CONFIG_NET_IPGRE is not set
# CONFIG_ARPD is not set
# CONFIG_INET_ECN is not set
# CONFIG_SYN_COOKIES is not set
# CONFIG_IPV6 is not set
# CONFIG_KHTTPD is not set
# CONFIG_ATM is not set
# CONFIG_VLAN_8021Q is not set
# CONFIG_IPX is not set
# CONFIG_ATALK is not set
# CONFIG_DECNET is not set
# CONFIG_BRIDGE is not set
# CONFIG_X25 is not set
# CONFIG_LAPB is not set
# CONFIG_LLC is not set
# CONFIG_NET_DIVERT is not set
# CONFIG_ECONET is not set
# CONFIG_WAN_ROUTER is not set
# CONFIG_NET_FASTROUTE is not set
# CONFIG_NET_HW_FLOWCONTROL is not set

#
# QoS and/or fair queueing
#
# CONFIG_NET_SCHED is not set
```

```
#
# ATA/IDE/MFM/RLL support
#
# CONFIG_IDE is not set
# CONFIG_BLK_DEV_IDE_MODES is not set
# CONFIG_BLK_DEV_HD is not set

#
# SCSI support
#
# CONFIG_SCSI is not set

#
# Network device support
#
CONFIG_NETDEVICES=y

#
# ARCnet devices
#
# CONFIG_ARCNET is not set
# CONFIG_DUMMY is not set
# CONFIG_BONDING is not set
# CONFIG_EQUALIZER is not set
# CONFIG_TUN is not set
# CONFIG_ETHERTAP is not set

#
# Ethernet (10 or 100Mbit)
#
CONFIG_NET_ETHERNET=y
# CONFIG_MACE is not set
# CONFIG_BMAC is not set
# CONFIG_GMAC is not set
# CONFIG_SUNLANCE is not set
# CONFIG_SUNBMAC is not set
# CONFIG_SUNQE is not set
# CONFIG_SUNGEM is not set
# CONFIG_NET_VENDOR_3COM is not set
# CONFIG_LANCE is not set
# CONFIG_NET_VENDOR_SMC is not set
# CONFIG_NET_VENDOR_RACAL is not set
# CONFIG_NET_ISA is not set
# CONFIG_NET_PCI is not set
# CONFIG_NET_POCKET is not set

#
# Ethernet (1000 Mbit)
#
# CONFIG_ACENIC is not set
# CONFIG_DL2K is not set
# CONFIG_MYRI_SBUS is not set
# CONFIG_NS83820 is not set
# CONFIG_HAMACHI is not set
# CONFIG_YELLOWFIN is not set
# CONFIG_SK98LIN is not set
```

```
# CONFIG_FDDI is not set
# CONFIG_HIPPI is not set
# CONFIG_PLIP is not set
CONFIG_PPP=m
CONFIG_PPP_MULTILINK=y
# CONFIG_PPP_FILTER is not set
CONFIG_PPP_ASYNC=m
CONFIG_PPP_SYNC_TTY=m
CONFIG_PPP_DEFLATE=m
CONFIG_PPP_BSDCOMP=m
CONFIG_PPPOE=m
# CONFIG_SLIP is not set

#
# Wireless LAN (non-hamradio)
#
# CONFIG_NET_RADIO is not set

#
# Token Ring devices
#
# CONFIG_TR is not set
# CONFIG_NET_FC is not set
# CONFIG_RCPCI is not set
# CONFIG_SHAPER is not set

#
# Wan interfaces
#
# CONFIG_WAN is not set

#
# Amateur Radio support
#
# CONFIG_HAMRADIO is not set

#
# IrDA (infrared) support
#
# CONFIG_IRDA is not set

#
# ISDN subsystem
#
# CONFIG_ISDN is not set

#
# Old CD-ROM drivers (not SCSI, not IDE)
#
# CONFIG_CD_NO_IDESCSI is not set

#
# Console drivers
#

#
# Frame-buffer support
```

```
#
# CONFIG_FB is not set

#
# Input core support
#
# CONFIG_INPUT is not set
# CONFIG_INPUT_KEYBDEV is not set
# CONFIG_INPUT_MOUSEDEV is not set
# CONFIG_INPUT_JOYDEV is not set
# CONFIG_INPUT_EVDEV is not set

#
# Macintosh device drivers
#

#
# Character devices
#
# CONFIG_VT is not set
CONFIG_SERIAL=y
CONFIG_SERIAL_CONSOLE=y
# CONFIG_SERIAL_EXTENDED is not set
# CONFIG_SERIAL_NONSTANDARD is not set
CONFIG_UNIX98_PTYS=y
CONFIG_UNIX98_PTY_COUNT=32

#
# I2C support
#
CONFIG_I2C=y
# CONFIG_I2C_ALGOBIT is not set
# CONFIG_I2C_ALGOPCF is not set
CONFIG_I2C_SIMPLE_BIT=y
# CONFIG_I2C_ALGO8XX is not set
# CONFIG_I2C_CHARDEV is not set
# CONFIG_I2C_PROC is not set

#
# Mice
#
# CONFIG_BUSMOUSE is not set
# CONFIG_MOUSE is not set

#
# Joysticks
#
# CONFIG_INPUT_GAMEPORT is not set
# CONFIG_QIC02_TAPE is not set

#
# Watchdog Cards
#
# CONFIG_WATCHDOG is not set
# CONFIG_INTEL_RNG is not set
# CONFIG_NVRAM is not set
# CONFIG_RTC is not set
```

```
CONFIG_DS1306_RTC=y
# CONFIG_DTLK is not set
# CONFIG_R3964 is not set
# CONFIG_APPLICOM is not set

#
# Ftape, the floppy tape device driver
#
# CONFIG_FTAPE is not set
# CONFIG_AGP is not set
# CONFIG_DRM is not set

#
# Multimedia devices
#
# CONFIG_VIDEO_DEV is not set

#
# File systems
#
# CONFIG_QUOTA is not set
# CONFIG_AUTOFS_FS is not set
# CONFIG_AUTOFS4_FS is not set
# CONFIG_REISERFS_FS is not set
# CONFIG_REISERFS_CHECK is not set
# CONFIG_REISERFS_PROC_INFO is not set
# CONFIG_ADFS_FS is not set
# CONFIG_ADFS_FS_RW is not set
# CONFIG_AFFS_FS is not set
# CONFIG_HFS_FS is not set
# CONFIG_BFS_FS is not set
# CONFIG_EXT3_FS is not set
# CONFIG_JBD is not set
# CONFIG_JBD_DEBUG is not set
# CONFIG_FAT_FS is not set
# CONFIG_MSDOS_FS is not set
# CONFIG_UMSDOS_FS is not set
# CONFIG_VFAT_FS is not set
# CONFIG_EFS_FS is not set
# CONFIG_JFFS_FS is not set
CONFIG_JFFS2_FS=y
CONFIG_JFFS2_FS_DEBUG=0
CONFIG_JFFS2_FS_NAND=y
# CONFIG_CRAMFS is not set
# CONFIG_TMPFS is not set
# CONFIG_RAMFS is not set
# CONFIG_ISO9660_FS is not set
# CONFIG_JOLIET is not set
# CONFIG_ZISOFS is not set
# CONFIG_MINIX_FS is not set
# CONFIG_VXFS_FS is not set
# CONFIG_NTFS_FS is not set
# CONFIG_NTFS_RW is not set
# CONFIG_HPFS_FS is not set
CONFIG_PROC_FS=y
# CONFIG_DEVFS_FS is not set
# CONFIG_DEVFS_MOUNT is not set
```

```
# CONFIG_DEVFS_DEBUG is not set
CONFIG_DEVPTS_FS=y
# CONFIG_QNX4FS_FS is not set
# CONFIG_QNX4FS_RW is not set
# CONFIG_ROMFS_FS is not set
CONFIG_EXT2_FS=y
# CONFIG_SYSV_FS is not set
# CONFIG_UDF_FS is not set
# CONFIG_UDF_RW is not set
# CONFIG_UFS_FS is not set
# CONFIG_UFS_FS_WRITE is not set

#
# Network File Systems
#
# CONFIG_CODA_FS is not set
# CONFIG_INTERMEZZO_FS is not set
CONFIG_NFS_FS=y
CONFIG_NFS_V3=y
# CONFIG_ROOT_NFS is not set
CONFIG_NFSD=y
CONFIG_NFSD_V3=y
CONFIG_SUNRPC=y
CONFIG_LOCKD=y
CONFIG_LOCKD_V4=y
CONFIG_SMB_FS=y
# CONFIG_SMB_NLS_DEFAULT is not set
# CONFIG_NCP_FS is not set
# CONFIG_NCPFS_PACKET_SIGNING is not set
# CONFIG_NCPFS_IOCTL_LOCKING is not set
# CONFIG_NCPFS_STRONG is not set
# CONFIG_NCPFS_NFS_NS is not set
# CONFIG_NCPFS_OS2_NS is not set
# CONFIG_NCPFS_SMALLDOS is not set
# CONFIG_NCPFS_NLS is not set
# CONFIG_NCPFS_EXTRAS is not set
# CONFIG_ZISOFS_FS is not set

#
# Partition Types
#
# CONFIG_PARTITION_ADVANCED is not set
CONFIG_MSDOS_PARTITION=y
CONFIG_SMB_NLS=y
CONFIG_NLS=y

#
# Native Language Support
#
CONFIG_NLS_DEFAULT="iso8859-1"
# CONFIG_NLS_CODEPAGE_437 is not set
# CONFIG_NLS_CODEPAGE_737 is not set
# CONFIG_NLS_CODEPAGE_775 is not set
# CONFIG_NLS_CODEPAGE_850 is not set
# CONFIG_NLS_CODEPAGE_852 is not set
# CONFIG_NLS_CODEPAGE_855 is not set
# CONFIG_NLS_CODEPAGE_857 is not set
```

```
# CONFIG_NLS_CODEPAGE_860 is not set
# CONFIG_NLS_CODEPAGE_861 is not set
# CONFIG_NLS_CODEPAGE_862 is not set
# CONFIG_NLS_CODEPAGE_863 is not set
# CONFIG_NLS_CODEPAGE_864 is not set
# CONFIG_NLS_CODEPAGE_865 is not set
# CONFIG_NLS_CODEPAGE_866 is not set
# CONFIG_NLS_CODEPAGE_869 is not set
# CONFIG_NLS_CODEPAGE_936 is not set
# CONFIG_NLS_CODEPAGE_950 is not set
# CONFIG_NLS_CODEPAGE_932 is not set
# CONFIG_NLS_CODEPAGE_949 is not set
# CONFIG_NLS_CODEPAGE_874 is not set
# CONFIG_NLS_ISO8859_8 is not set
# CONFIG_NLS_CODEPAGE_1250 is not set
# CONFIG_NLS_CODEPAGE_1251 is not set
CONFIG_NLS_ISO8859_1=m
# CONFIG_NLS_ISO8859_2 is not set
# CONFIG_NLS_ISO8859_3 is not set
# CONFIG_NLS_ISO8859_4 is not set
# CONFIG_NLS_ISO8859_5 is not set
# CONFIG_NLS_ISO8859_6 is not set
# CONFIG_NLS_ISO8859_7 is not set
# CONFIG_NLS_ISO8859_9 is not set
# CONFIG_NLS_ISO8859_13 is not set
# CONFIG_NLS_ISO8859_14 is not set
# CONFIG_NLS_ISO8859_15 is not set
# CONFIG_NLS_KOI8_R is not set
# CONFIG_NLS_KOI8_U is not set
# CONFIG_NLS_UTF8 is not set

#
# Sound
#
# CONFIG_SOUND is not set

#
# MPC8xx CPM Options
#
# CONFIG_SCC_ENET is not set
CONFIG_FEC_ENET=y
CONFIG_USE_MDIO=y
# CONFIG_FEC_AMD79C874 is not set
# CONFIG_FEC_LXT970 is not set
CONFIG_FEC_LXT971=y
# CONFIG_FEC_QS6612 is not set
CONFIG_FEC_KS8737=y
CONFIG_FEC_BCM5221=y
CONFIG_ENET_BIG_BUFFERS=y
CONFIG_SMC1_UART=y
CONFIG_CONS_SMC1=y
CONFIG_UART_MAXIDL_SMC1=1
CONFIG_SMC1_UART_RX_BDNUM=2
CONFIG_SMC1_UART_RX_BDSIZE=8
CONFIG_SMC1_UART_TX_BDNUM=2
CONFIG_SMC1_UART_TX_BDSIZE=8
# CONFIG_SMC2_UART is not set
```

```
CONFIG_USE_SCC_IO=y
CONFIG_SCC1_UART=y
CONFIG_PORT_CTS1_NONE=y
# CONFIG_UART_CTS_CONTROL_SCC1 is not set
CONFIG_PORT_RTS1_NONE=y
# CONFIG_PORT_RTS1_B is not set
# CONFIG_PORT_RTS1_C is not set
CONFIG_PORT_CD1_NONE=y
# CONFIG_UART_CD_CONTROL_SCC1 is not set
CONFIG_PORT_DTR1_NONE=y
# CONFIG_PORT_DTR1_A is not set
# CONFIG_PORT_DTR1_B is not set
# CONFIG_PORT_DTR1_C is not set
# CONFIG_PORT_DTR1_D is not set
CONFIG_UART_MAXIDL_SCC1=4
CONFIG_SCC1_UART_RX_BDNUM=2
CONFIG_SCC1_UART_RX_BDSIZE=8
CONFIG_SCC1_UART_TX_BDNUM=2
CONFIG_SCC1_UART_TX_BDSIZE=8
# CONFIG_SCC2_UART is not set
# CONFIG_SCC3_UART is not set
# CONFIG_SCC4_UART is not set
CONFIG_8xx_COPYBACK=y
# CONFIG_8xx_CPU6 is not set
# CONFIG_UCODE_PATCH is not set

#
# USB support
#
# CONFIG_USB is not set
# CONFIG_USB_UHCI is not set
# CONFIG_USB_UHCI_ALT is not set
# CONFIG_USB_OHCI is not set
# CONFIG_USB_AUDIO is not set
# CONFIG_USB_BLUETOOTH is not set
# CONFIG_USB_STORAGE is not set
# CONFIG_USB_STORAGE_DEBUG is not set
# CONFIG_USB_STORAGE_DATAFAB is not set
# CONFIG_USB_STORAGE_FREECOM is not set
# CONFIG_USB_STORAGE_ISD200 is not set
# CONFIG_USB_STORAGE_DPCM is not set
# CONFIG_USB_STORAGE_HP8200e is not set
# CONFIG_USB_STORAGE_SDDR09 is not set
# CONFIG_USB_STORAGE_JUMPSHOT is not set
# CONFIG_USB_ACM is not set
# CONFIG_USB_PRINTER is not set
# CONFIG_USB_DC2XX is not set
# CONFIG_USB_MDC800 is not set
# CONFIG_USB_SCANNER is not set
# CONFIG_USB_MICROTEK is not set
# CONFIG_USB_HPUSBSCSI is not set
# CONFIG_USB_PEGASUS is not set
# CONFIG_USB_KAWETH is not set
# CONFIG_USB_CATC is not set
# CONFIG_USB_CDCETHER is not set
# CONFIG_USB_USBNET is not set
# CONFIG_USB_USS720 is not set
```

```
#
# USB Serial Converter support
#
# CONFIG_USB_SERIAL is not set
# CONFIG_USB_SERIAL_GENERIC is not set
# CONFIG_USB_SERIAL_BELKIN is not set
# CONFIG_USB_SERIAL_WHITEHEAT is not set
# CONFIG_USB_SERIAL_DIGI_ACCELEPORT is not set
# CONFIG_USB_SERIAL_EMPEG is not set
# CONFIG_USB_SERIAL_FTDI_SIO is not set
# CONFIG_USB_SERIAL_VISOR is not set
# CONFIG_USB_SERIAL_IPAQ is not set
# CONFIG_USB_SERIAL_IR is not set
# CONFIG_USB_SERIAL_EDGEPORT is not set
# CONFIG_USB_SERIAL_KEYSPAN_PDA is not set
# CONFIG_USB_SERIAL_KEYSPAN is not set
# CONFIG_USB_SERIAL_KEYSPAN_USA28 is not set
# CONFIG_USB_SERIAL_KEYSPAN_USA28X is not set
# CONFIG_USB_SERIAL_KEYSPAN_USA28XA is not set
# CONFIG_USB_SERIAL_KEYSPAN_USA28XB is not set
# CONFIG_USB_SERIAL_KEYSPAN_USA19 is not set
# CONFIG_USB_SERIAL_KEYSPAN_USA18X is not set
# CONFIG_USB_SERIAL_KEYSPAN_USA19W is not set
# CONFIG_USB_SERIAL_KEYSPAN_USA49W is not set
# CONFIG_USB_SERIAL_MCT_U232 is not set
# CONFIG_USB_SERIAL_KLSI is not set
# CONFIG_USB_SERIAL_PL2303 is not set
# CONFIG_USB_SERIAL_CYBERJACK is not set
# CONFIG_USB_SERIAL_XIRCOM is not set
# CONFIG_USB_SERIAL_OMNINET is not set
# CONFIG_USB_RIO500 is not set

#
# Bluetooth support
#
# CONFIG_BLUEZ is not set

#
# Kernel hacking
#
# CONFIG_MAGIC_SYSRQ is not set
# CONFIG_KGDB is not set
# CONFIG_XMON is not set

#
# Library routines
#
CONFIG_ZLIB_INFLATE=y
CONFIG_ZLIB_DEFLATE=y
```

APPENDIX B

DNP3 PROTOCOL

DNP3 is a SCADA protocol designed to allow SCADA devices to communicate and transfer data and control commands from one point to another. It supports both serial and TCP/IP communications with IP communications generally being achieved by tunneling the serial version inside TCP or UDP packets. In DNP3 the term *outstation* refers to devices or computers that are in the field and the term *master* refers to computers in the control center. The terms *slave* is also used to refer to an outstation. DNP3 is a non-proprietary protocol and a full specification is available from www.dnp.org for a nominal fee.

Every DNP3 device has a database of different data types; each organized as an array of values. Data items are identified by their data type and index, called a point index, as shown in figure B.1. The MTU, or master, uses the data values to display the state or condition of the physical system to which one or more outstations or RTUs are connected. The objective of the master is to keep its database updated and accomplishes this by sending requests (polling) to outstations. Outstations then provide the master with the value of data item or items that were requested. (Outstations may also send unsolicited data to masters in the form of an unsolicited response).

The DNP3 protocol is organized into layers that are similar to the standard OSI model. The top layer, or user layer, maps DNP3 data objects to local data or a local

database. On master stations the user layer initiates requests to outstations for data and on outstations the user layer retrieves data from the outstation database in response to a master's request. The requested operations, the item or items on which the request is made, and any data need to complete the request are specified in the application layer message. The transport layer breaks up long application layer messages into smaller packets for the link layer and reassembles them on the other side. And the link layer makes the physical connection reliable. For the purpose of this disertation only the application layer is of concern since the developed security enhancements are applied at the application layer.

DNP3 application layer messages have three main components: an application header, a function code, and one or more DNP3 objects each of which consists of an object header followed by one or more object values. Figure B.2 shows the general structure of the DNP3 application layer fragments; masters send request fragments and outstations send response fragments. The application control octet contains application layer sequence number and some status flags. The function code indicates the operations being performed, and the object header identifies the data point or points (range of indexes) on which the operation is to be carried out. The IIN in the application response is used to communicate the internal status of the RTU or outstation to the master.

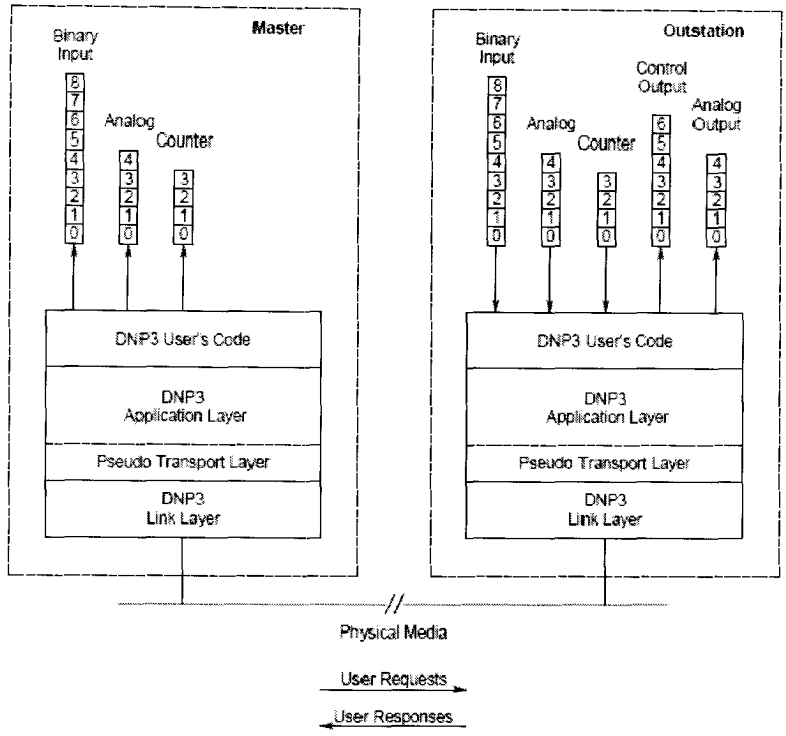


Figure B.1. DNP3 protocol layers.

Request Fragment

Application Request Header		First Object Header			DNP3 Objects		Last Object Header			DNP3 Objects
Application Control	Function Code	Group	Variation	Range			Group	Variation	Range	

Response Fragment

Application Response Header			First Object Header			DNP3 Objects		Last Object Header			DNP3 Objects
Application Control	IIN	Function Code	Group	Variation	Range			Group	Variation	Range	

Figure B.2. DNP3 application layer fragments.

Security in DNP3 is limited to transmission errors which are detected by the link layer using CRC checksums. As is common in many SCADA protocols the identity of the sender is not authenticated nor is the integrity of the message verified. More importantly, like other SCADA protocols DNP3 has embraced TCP/IP and the use of TCP and UDP are now apart of the DNP3 specification. The lack of security features in

DNP3 makes SCADA and DCS devices vulnerable to spoofing and replay attacks discussed in chapter two.

APPENDIX C

NESSUS SCAN REPORT

```
Nessus Scan Report
------------------


SUMMARY

 - Number of hosts which were alive during the test : 1
 - Number of security holes found : 1
 - Number of security warnings found : 2
 - Number of security notes found : 5


TESTED HOSTS

 10.165.49.249 (Security holes found)


DETAILS

+ 10.165.49.249 :
 . List of open ports :
   o ftp (21/tcp) (Security notes found)
   o telnet (23/tcp) (Security notes found)
   o general/tcp (Security warnings found)
   o general/udp (Security hole found)
   o general/icmp (Security warnings found)

 . Information found on port ftp (21/tcp)


    The service closed the connection after 0 seconds without sending
any data
    It might be protected by some TCP wrapper


 . Information found on port telnet (23/tcp)


    The service closed the connection after 0 seconds without sending
any data
    It might be protected by some TCP wrapper
```

. Warning found on port general/tcp

The remote host does not discard TCP SYN packets which
have the FIN flag set.

Depending on the kind of firewall you are using, an
attacker may use this flaw to bypass its rules.

See also : http://archives.neohapsis.com/archives/bugtraq/2002-10/0266.html
http://www.kb.cert.org/vuls/id/464113

Solution : Contact your vendor for a patch
Risk factor : Medium
BID : 7487

. Information found on port general/tcp

The remote host is running one of these operating systems :
Linux Kernel 2.4
NetGear Router

. Information found on port general/tcp

10.165.49.249 resolves as private049249.private.louisville.edu.

. Vulnerability found on port general/udp :

It was possible
to make the remote server crash
using the 'nestea' attack.

An attacker may use this flaw to
shut down this server, thus
preventing your network from
working properly

Solution : contact your operating
system vendor for a patch.

Risk factor : High
CVE : CAN-1999-0257
BID : 7219

. Information found on port general/udp

For your information, here is the traceroute to 10.165.49.249 :
136.165.67.244
136.165.67.254

10.165.49.249

. Warning found on port general/icmp

The remote host answers to an ICMP timestamp request. This allows an attacker to know the date which is set on your machine.

This may help him to defeat all your time based authentication protocols.

Solution : filter out the ICMP timestamp requests (13), and the outgoing ICMP timestamp replies (14).

Risk factor : Low
CVE : CAN-1999-0524

This file was generated by the Nessus Security Scanner

APPENDIX D

SIXNET mIPM TECHNICAL SPECIFICATIONS

Section 5 Technical Specifications

Technical Specifications

Here are the technical specifications for the VersaTRAK mIPm RTU / Controller.

General	Industrial PowerPC (32 bit data bus)
Compatibility with legacy Mini VersaTRAK RTU	Fully compatible with an assortment of on board I/O
Local I/O (on board)	26
Operating system	Embedded LINUX
Unique station addresses (unit Ids)	16,000+ (SIXNET) or 247 (Modbus)
Dynamic memory (RAM) (for program execution, dynamic variables, dynamic file system, etc.)	32bit, 0 wait states 16 Megabytes
Program memory (Flash) (for Linux OS, program storage, and file system)	16 Megabytes
Datalogging memory (RAM) (for datalogging and retained variables)	Battery-backed – Rechargeable Lithium 512K bytes
Battery-backup time / life	1 year / 10+ years
Real-time clock resolution	10 mS
Real-time clock accuracy	+/-15 seconds per month
I/O expansion	EtherTRAK, RemoteTRAK
Maximum distributed I/O	256 physical I/O
Maximum virtual I/O registers	256 per I/O type
Datalogging support	Yes – SIXNET Sixlog
Datalogging modes	Trending, alarm logging, sequence of events, event initiated, client transfers, and others
LINUX capabilities	Practically unlimited
Programming	High level C and others
Number of applications allowed	As many as there is memory for
Available FREE source code	Practically unlimited
IEC 61131 PLCopen programming	Yes – SIXNET ISaGRAF
Languages supported	Ladder logic, function chart, function block, instruction list, structured text, and flow chart
Cycle time	10 mS minimum (user settable)
Communication capabilities	Master, slave, peer to peer, report on exception, store and forward and more
Communication media supported	Ethernet, telemetry, telephone (dialup and leased line), radio (dumb and smart), other wireless, fiber optic, short haul and more

Watchdogs and Monitors	For run-time diagnostics
CPU watchdog	CPU automatically resets if error is detected; status LED flashes error code
Communications watchdog	Settable timeout and output action (force off or freeze)
Heartbeat watchdog	Settable timeout & output action (force off or freeze)
Ethernet Port(s)	10/100Base1x (auto-detecting)
Connection	RJ45 (auto-crossover)
Isolation	1500 Volts RMS 1 minute (60 Hz)
Message response time (typical)	5 mS
Diagnostic LEDs	Indicates speed and activity
Protocols supported	TCP/IP, ARP, UDP, ICMP, DHCP, Modbus/TCP, SIXNET, and more
Network port	1 shielded RJ45 connector
Serial Ports	300 to 115,200 baud
RS232 Port B	RJ45 (TD, RD, CTS, RTS, CD, DTR, DSR/RI , GND)
RS232 Port D	RJ45 (TD, RD, RTS, GND)
RS485 Port A, C	Screws (GND, 485+, 485-, termination) (2-wire half-duplex) (Port C GND common with port D)
RS485 network	Up to 32 (full-load) stations
RS485 distance	Up to 0.5 miles (1 km)
Protocols (master & slave)	SIXNET & Modbus (RTU and ASCII); Many others available in LINUX
Diagnostic LEDs on each port	Transmit Data (TD) & Receive Data (RD)
Flow Control	Hardware, software, RTS-party (for radios and RS485)
Discrete Inputs	12 Channels
Guaranteed ON Voltage	9 VDC
Maximum Voltage	30 VDC
Guaranteed OFF	5.0 VDC & 1.5mA DC
Input Resistance	10K Ohms
Input Current @ 24V	3 mA
Filtered ON/OFF delay	25ms (20Hz max. counting)
Fast ON/OFF delay	4 ms (100Hz max counting)
Count Rate	(10khz on channel 1, see above for other rates)
Discrete Outputs	4 or 8 channels (10-30VDC)
Maximum Output per channel	1 Amp
Maximum Output per module	8 Amps
Max. OFF state leakage	0.05 mA
Minimum Load	1 mA
Inrush current	5 Amps (100 ms surge)
Typical ON resistance	0.3 ohms
Typical ON voltage (@1A)	0.3 VDC
Analog Inputs	6 or 8 channels (4-20 mA)
A/D resolution	16 bits (0.003%)
Full scale accuracy	+/- 0.1% (@20°C)
Span and offset temp. coef.	+/- 50 ppm per degree C
Input impedance	100 Ohm
Current protection	Self-resetting fuses
DMRR (differential mode rejection)	66 dB at 50/60 Hz
Analog Outputs	Up to 2 channels (4-20mA)
D/A resolution	16 bits (less than 1µA)
Full Scale accuracy	+/- 0.02%
Span and offset temp. coef.	+/- 50 ppm per °C typical
Max. output settling time	5 ms (to .05%)
Load resistance range (@ +24 VDC supply	0-750 Ohms
Short circuit protection	Current limiting

I/O Tool Kit Windows Software	Level 1 provided free with all systems
Operating systems	95, 98, ME, NT, 2000, and XP
Minimum system requirements	Pentium or equivalent, 32 Mb RAM, 100 Mb hard disk space
Option 1 (Basic)	Configuration, calibration, diagnostics, and limited exporting of I/O definitions.
Option 2 (SCS)	Full importing and exporting of I/O definitions, peer to peer I/O transfers, and quick load feature "Load All".
Option 3 (Datalogging)	Datalogging capability and datalog server
Option 4 (LINUX)	LINUX functionality and support.
Environmental	DIN rail or flat panel mount
Input voltage	10-30 VDC (integrated switching supply). (External AC/DC or DC/DC supplies optional)
Input power (@ 24 VDC unless otherwise noted) (+/-10%) (Note: The power consumption variations mostly depend on the number of Ethernet and/or serial connections.)	VT-IPm: 2.4 W (100mA) – typical (no communications)
Temperature	-40 to 70°C (-40 to 85°C storage)
Humidity	5% to 95% RH (non-condensing) (optional conformal coating)
Flammability	UL 94V-0 materials
Electrical Safety	UL 508, CSA C22.2/14; EN61010-1 (IEC1010); CE
EMI emissions	FCC part 15, ICES-003, Class A; EN55022; EN61326-1; CE
EMC immunity	EN61326-1 (EN61000-4-2,3,4,6); CE
Surge withstand	IEEE-472 (ANSI C37.90); EN61000-4-2, 4
Vibration	IEC68-2-6
Hazardous locations (Class 1, Div 2, Groups A, B, C, D)	UL 1604, CSA C22.2/213, Cenelec EN50021 Zone 2
Marine & Offshore	DNV (Det Norske Veritas)
Packaging	Lexan Packaging
Mounting	DIN rail (EN50022) or direct to panel
Size	4.75"L x 3.83"W x 4.13"H (12.07cm L x 9.73cm W x 10.48cm H)

CURRICULUM VITAE

Jeffrey Lloyd Hieb
497 Sacree Rd, Shelbyville, KY 40065 (502) 418 - 6106
jlhieb01@insightbb.com

Education

Ph.D., Computer Science and Engineering, expected spring 2008.
University of Louisville
Louisville, KY
Dissertation advisor: Professor James H. Graham
Dissertation: Security-Enhanced Remote Terminal Units for SCADA Networks

M.S., Computer Engineering and Computer Science, December 2004
GPA: 3.8/4.0
University of Louisville
Louisville, KY
Thesis advisor: Professor James H. Graham
Thesis: Anomaly Based Intrusion Detection for Intrusion Monitoring Using a Dynamic Honeypot.

B.S. Computer Science / B.A. Philosophy June 1992
Cum Laude
Furman University
Greenville, SC

Experience

Graduate Research Assistant
January 2005 – present
Intelligent Systems Research Lab, University of Louisville Louisville, KY
Plant Manager / Plant Supervisor
June 1992 – July 2004
Hieb Concrete Products Inc. Shelbyville, KY
Lighting and Sound Technician
October, 1988- June 1992
McAlistar Auditorium, Furman University Greenville, SC

Honors

Phi Beta Kappa,
Furman University, 1992
Upsilon Pi Epsilon,
University of Louisville, Fall 2004
CECS Master of Science, highest cumulative scholastic standing
University of Louisville, April 2005

Professional Organizations

Member, Association for Computing Machinery (ACM).
Member, Institute of Electrical and Electronics Engineers (IEEE).

Publications

P. A. Ralston, J. H. Graham, and J. L. Hieb, "Cyber Security Risk assessment for SCADA and DCS Networks" *ISA Transactions*, vol. 46, no. 4, pp. 583-594, 2007.

J. L. Hieb, S. C. Patel, and J. H. Graham, "Security Hardened Remote Terminal Units," in *Critical Infrastructure Protection: Issues and solutions*, S. Shenoi and E. Goetz, Eds. Boston: Springer, 2007.

J. L. Hieb, P. A. Ralston, and J. H. Graham, "Testing Security Enhanced DNP3 on Actual SCADA System," in *Proceedings of the International Telecommunications Education Research Association (ITERA 2007)*, Louisville, KY, March 2007 pp. 354-378.

J. H. Graham, M. S. Mostafa, B. Arazi, A. Tantawy, J. L. Hieb, P. A. Ralston, and S. C. Patel, "Improvements in SCADA and DCS Systems Security," in *Proceedings of the ISCA 22nd International Conference on Computers And Their Applications (CATA 2007)*, Honolulu, HI, March 2007.

J. L. Hieb and J. H. Graham, "Security Enhanced Remote Terminal Units for SCADA Networks," in *Proceedings of ISCA 19th International Conference on Computers Applications in Industry and Engineering (CAINE 2006)* Las Vegas, NV, November 2006, pp. 271-276.

J. L. Hieb and J. H. Graham, "Dynamic Authentication Using Keystroke Biometrics," in *Proceedings of the 21st International Conference on Computers and Their Applications (CATA 2006)*, Seattle, WA, March 2006, pp. 337-342.

J. L. Hieb and J. H. Graham, "Anomaly-based Intrusion Monitoring Using A Dynamic Honeypot," in *Proceedings of the 20th International Conference on Computers and Their Applications (CATA 2005)*, March 16-18, New Orleans, 2005.

J. L. Hieb, "Improving the SmallTalk Browser: A Case Study in SmallTalk Development," in *Proceedings of 28th Annual Southeast Regional Conference of ACM.* Greenville, SC, April 18-20, 1990, pp. 242-247.

Presentations

"Security Hardening Remote Terminal Units", *First IFIP WG 11.10 International Conference on Critical Infrastructure Protection*, Dartmouth College, Hanover New Hampshire, March 18-21, 2007.