Cover Page

Security Issues in the Architecture of the Global Command and Control System (GCCS)

Experience Paper

Shawn A. Butler

15 July, 1998

(GCCS Chief Systems Engineer, 1994 – 1995)

Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania
E-mail shawnb@cs.cmu.edu
Phone: 412-683-6555
FAX: 412-683-5275

Key Words: Architecture, Security, Command and Control, Common Operating Environment, COE, GCCS,

ABSTRACT
The Global Command and Control System (GCCS) was one of the most ambitious and largest software integration tasks in the history of the Department of Defense. As the Chief Systems Engineer for GCCS, I found architectural differences among command and control systems presented unique integration and interoperability challenges. In this paper I present 3 security-related examples of specific problems I encountered when I attempted to integrate several systems into GCCS. I also discuss the problem of system-level security analysis and introduce a framework that software engineers can use to evaluate security.

Security Issues in the Architecture of the Global Command and Control System (GCCS)

Experience Paper

Shawn A. Butler
15 July, 1998
(GCCS Chief Systems Engineer, 1994 – 1995)

Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania

ABSTRACT
The Global Command and Control System (GCCS) was one of the most ambitious and largest software integration tasks in the history of the Department of Defense. As the Chief Systems Engineer for GCCS, I found architectural differences among command and control systems presented unique integration and interoperability challenges. In this paper I present 3 security-related examples of specific problems I encountered when I attempted to integrate several systems into GCCS. I also discuss the problem of system-level security analysis and introduce a framework that software engineers can use to evaluate security.

1. Introduction

The Global Command and Control System (GCCS) was one of the most ambitious and largest software integration tasks in the history of the Department of Defense. Applications in all stages of maturity were chosen to be integrated into a seamless system, organized around the Common Operating Environment (COE). The COE was a collection of software components commonly found in all command and control systems. As the Chief Systems Engineer for GCCS, I was responsible for every aspect of integration and development including GCCS security.

Security proved the most difficult of all the system integration tasks for two reasons. First, although security specialists talked about the "security architecture" of GCCS, a security checklist derived from a set of security requirements and policies was the best they could produce. Checklists provide a piecemeal approach to system security and usually lack a system level perspective. GCCS interoperability requirements and the process of integrating legacy applications highlighted the role that architectures and system designs played in GCCS security. Second, user's demands for configuration flexibility presented significant challenges to maintaining a consistent level of security with each system. A team of independent security specialists verified the system's security just before fielding. Each security evaluation drained off scarce resources for several weeks at a time. The security team attempted to find security flaws using whatever means they considered reasonable. System security was re-verified each time the configuration of GCCS changed, which was almost monthly during initial fielding.

The Department of Defense relies on a security process that is not compatible with modern software development processes and designs. What I really needed were concrete architectural and design guidance and methodologies for analyzing system security that did not depend on a security specialist's ability to defeat the system after I build it. My frustrations with these two problems led to my current research and the beginnings of a framework to help solve the second problem.

2. Background

For many years the Department of Defense operated the World Wide Military Command and Control System (WWMCCS) as the primary command and control system[1]. WWMCCS was a distributed information system that linked major military command centers throughout the world, such as the European and Pacific theaters and the National Military Command Center in the Pentagon. The system processed TOP SECRET, SECRET, and UNCLASSIFIED information, but the bulk of information was SECRET. Since WWMCCS did not have multi-level security, the system operated as if all the information were TOP SECRET. The security requirements for a TOP SECRET system are greater than for systems processing SECRET information.

Military computer security requirements are found in a number of military directives, regulations, and publications. The most well known set of publications are the "rainbow" series, which consist of more than 20 books, each book a different color. The Orange Book defines the concept of a Trusted Computing Base (TCB) and specifies the TCB requirements for increasing levels of security. UNIX systems are evaluated and classified based on the criteria established in the Orange Book. Ordinary UNIX systems usually fall into the C1 or C2 class, which is characterized by discretionary security protection requirements. Operating systems classified at the B or A level meet increasingly stricter security requirements and are usually highly specialized operating systems.

The system consisted of 40 Honeywell mainframe computers that serviced numerous dumb terminals within each major command center and in isolated locations throughout the world. Initially built during the 1970's, WWMCCS had become quickly outdated so a modernization program was initiated during the early 1980's [WWMCCS92]. Research, development, test, and evaluation for the modernization program was budgeted for $773 million, By 1987 the program was behind schedule and over budget so congress cut the FY 88 budget to $21 million. Technology rapidly passed the WWMCCS system and users became increasingly dissatisfied with WWMCCS capabilities. By the mid-nineties most other command and control systems had far exceeded WWMCCS functionality. However, none of the newly developed command and control systems could meet the WWMCCS user's functional requirements.

3. GCCS

The Global Command and Control System, a highly distributed client/server system, was conceived as the replacement for WWMCCS. The initial version of GCCS

[1] "Command and control" is a term used to define the activity of monitoring, planning and directing military resources.

was a conglomeration of existing command and control applications and new applications that increased and replaced WWMCCS functionality. GCCS consisted of two parts: the Common Operating Environment (COE) and the Application Layer. In order to keep development and fielding costs to a minimum, GCCS consisted of commercial hardware and software and processed only SECRET information. Not only did this simplify the security requirements, but this also meant that GCCS could be fielded on standard commercial UNIX operating systems instead of more secure, and very expensive B2 operating systems. I was responsible for mitigating the risks associated with security weaknesses in the UNIX operating system.

Although most major system development efforts take 5 to 10 years, the Joint Chiefs of Staff wanted the replacement system within 2-3 years beginning in 1994. The primary motivation for the rapid development cycle was the enormous cost of operating WWMCCS, estimated at $7,000,000/month. The 2-3 year development constraint was thought attainable for several reasons. First, GCCS was to be built using existing applications, therefore, GCCS was simply considered an integration exercise, rather than new development. I believe there is a general misconception that integration efforts take less time than new development. Stakeholders assumed that most of the applications selected to be part of GCCS fulfilled enough of the user's requirements that little or no additional development needed to be done. Applications were selected from various Department of Defense agencies and services based on how well they met user requirements and other factors, the least of which was the ease with which they could be integrated, maintained, scaled, or extended.

3.1 GCCS Architecture

The foundation of GCCS is the Common Operating Environment (COE), 18 abstract functional components that, when implemented, form the infrastructure services and a set of standard components for all GCCS applications. All existing or legacy applications had to "migrate" to the GCCS COE. Migration required applications compliance with engineering guidance in 4 areas: integration and run-time, user interface, architecture, and software quality. Software for the COE came from each of the services, and the Defense Mapping Agency. I was charged with integrating the COE components and more than 20 legacy applications, all in various stages of development, into a single command and control system that could be uniquely configured at each operational site. GCCS was really a set of command and control applications, which any site could install components as needed.

COE components fall into 3 categories: the kernel, infrastructure services, and common support application components (figure 1). Kernel components consist of the operating system, window libraries (X11R5 and Motif), printing service, executive manager, name service, and a security/system management service. Kernel components are considered essential system components, i.e. every workstation requires these services regardless of function. The security service provides tools to allow system administrators to set up various types of access control accounts. The kernel configuration is tightly controlled since slight deviations from the established configuration could cause disastrous system integration problems. All application developers are expected to develop to the kernel configuration and each developer receives a copy of the kernel and a set of tools to ensure that they follow the run-time integration engineering guidelines.

The infrastructure services provide the middleware for the applications. The middleware consists of the following components: management services, communication services, distributed computing services, presentation and web services and data and object management services. Management services are network and system management tools that system administrators use to monitor the system. Distributed Computing Environment (DCE) provided the distributed computing service and the Common Object Request Broker Architecture (CORBA) served as the data and object management service, although the initial GCCS fielded system did not use either service. The communication service provides the interface to external systems. Most external interfaces consisted of messages sent to and from GCCS. Netscape and Internet Rely Chat implemented the web services, but the presentation service was not specified at the time.

The Common Support Layer of the COE consisted of the group of applications that are common to all command and control systems such as office automation applications, situation displays, message generation and management software, etc. At the time, office automation applications, such as word processors, spreadsheets, and slide presentation software did not compare to the products used on personnel computers. UNIX based office automation software had considerably less functionality than PC products. The biggest drawback to the UNIX software was the incompatibility of file formats. Users had hundreds of Microsoft Powerpoint files that were not exportable to the UNIX office automation software and any files created on the UNIX system were not exportable to the PC system. Although PC emulators could have provided a temporary fix to the office automation problem they were too expensive.

3.2 Interoperability Issues

The primary drawback of the existing command and control systems was their lack of interoperability between services. Since joint military operations nearly always consist of units from the Marine Corps, Navy, Air Force and Army working together, joint military operations require a command and control system that is interoperable among the other service's command and control systems. As an illustration, many of the frustrations experienced during Desert Storm occurred because sytems were not interoperable. Information was frequently exchanged using floppy disks or paper printouts which then had to be re-keyed into an electronic form. As a consequence of the experiences in Desert Storm, interoperability became the number one command and control system requirement in the Department of Defense.

Although interoperability was a critical requirement in joint operations, it was not well defined. Interoperability meant different things to different users and under different circumstances. Ideally, systems should be able to efficiently exchange data without any loss of meaning or content, but in practice this is very difficult. The Department of Defense outlines 4 levels of interoperability for command and control systems. The highest level "is characterized by the ability to globally share integrated information in a distributed information space."[I&RTS96]. Level 4 was the ultimate goal for GCCS, but each application implemented lower levels of interoperability.

In some cases, application portability across different hardware platforms or operating systems was sufficient to meet interoperability requirements. Data is exchanged because operators from different services are co-located. Each service purchased their own computer hardware so applications built to run on Sun Microsystem hardware did not have to be converted to the Hewlett Packard hardware and vice versa. The lack of portability forced users of one service to learn the other service's application, or for the application and hardware system to integrate with the larger system.

Interoperability could also be achieved if systems could interface using formatted messages, e-mail, or import/export functions. In practice this method was flawed. Currently, all command and control systems communicate with other systems using standard message sets. Unfortunately, the "standard" message sets are not truly standard and not particularly efficient for transmitting all types of information. Many of the message standards were developed before multi-media applications became integrated into command and control systems. Each command and control system selected from several standard message sets, which meant that each command and control system used a different set. In addition, many of the sets were extended with unique messages that were not compatible with the DoD standard. Interoperability through messages design limits interoperability for two reasons: 1) users are limited by message content and 2) the information is only available when it is sent.

A common view of the battlefield is essential to effective military operations. A higher level of interoperability is required when users shared information from the same source. A common view is ensured when all users have access to the same information source. In practice, different database schemas and data elements made it nearly impossible to share information from a central location. In my experience, integrating databases is one of the most difficult engineering tasks, however, it also provides the greatest interoperability.

3.3 Architectural Security Issues and Interoperability

3.3.1 Interoperability incompatibilities

However interoperability was achieved between two systems, there were usually security implications. If messages were exchanged then encryption of the messages as they pass between two systems was usually sufficient to control access to the information. Encryption incurs maintenance costs because the DoD relies on special hardware for all encryption. The DoD builds many types of encryption devices, all of which are incompatible with each other. No matter which encryption device is chosen, the hardware is scarce and not compatible with other systems that use different encryption devices. Incompatible encryption components make interoperability nearly impossible. This detail is often overlooked when designing command and control systems. However, when two systems share a common database, then access controls to the database become a primary security concern and incompatibilities between systems can surface. For instance, two applications required access to classified data in the database. One application used database access control mechanisms to ensure that unauthorized personnel did not get unlimited access to the data. Users were restricted from viewing or writing to particular rows, or restricted from certain tables in the database. The other application restricted a user's access to the data by controlling access to the application. Implicit in the latter design is an assumption that any user with access to the application has unlimited access to the database. These two fundamentally different, but valid, points of access control made integration of these applications into a seamless system difficult.

3.3.2 Additional integration problems

GCCS interoperability requirements, integration of legacy applications and the user's demand for configuration flexibility presented significant challenges to maintaining a consistent level of security with each system. Some other security integration problems with the GCS architecture were access control designs and application programming (API) interface mismatches.

Access control designs of two systems created a particularly difficult problem. Access controls were usually based on an operator's role or position and the role could change during the operator's shift or an operator may have several roles during the same shift. Problems arose when one system required an operator to log out and then log in when he switched roles, in effect restricting operators from assuming two roles simultaneously. Although this simplified audit trails in that system, it was an unacceptable specification in another system. Security administrators needed the flexibility to accommodate both requirements. Eventually a scheme for access control was developed that was acceptable to all users.

A third problem arose when we discovered incompatibilities between security technologies. Specifically, the Fortezza system developed by the National Security Agency (NSA) was incompatible with Kerberos. Fortezza, NSA's smart card technology, was the latest security mechanism that promised improved system security.

NSA considered Kerberos inadequate for GCCS and insisted that GCCS implement the Fortezza system. Although Kerberos had recognized flaws it was available and used in commercial systems. Fortezza didn't have Kerberos' flaws but wasn't available in production quantities.

Furthermore, NSA had not yet developed a Fortezza card that had been adequately tested for SECRET systems. The initial GCCS design used Kerberos and later integrated Fortezza when it became available. Unfortunately, incompatibilities between the application programming interfaces (API's) surfaced, and made integration of the two technologies impossible until the API conflicts were resolved. NSA quickly began to work with members of the Open Systems Foundation, however, the process was expected to take at least two years.

3.4 Architectural integration summary

As the GCCS chief engineer, it was obvious to me that the security of a system does not depend solely on a collection of "silver bullet" technologies and checklists. I could not integrate two systems and plan to overlay the security later. The system security must be designed hand in hand with the system architecture. Interoperability requirements and legacy system integration concerns are not confined to the Department of Defense. As commercial organizations expand and grow so do their interoperability requirements. Companies such as SAP specialize in integrating reusable components. Common system engineering questions include the following: What are the design principles and engineering guidance that system engineers should follow? How does the architecture support system security? What security mechanisms are appropriate for a particular architectural style? What are the security weaknesses associated with an architectural style? What security conflicts should system engineers look for? What are the design pitfalls? How do interoperability requirements affect security? The list could go on but answering any of these questions would be extremely useful to system developers.

4. Security implementation

In addition to the architectural issues of integration and interoperability, I was overwhelmed with the myriad security technologies and designs available at the time. While some security solutions were dictated by regulations, I retained a great deal of flexibility to select the mechanisms that constituted the system's security. Frequently, the tension between performance and maintainability and security, raises such questions as: Since GCCS is unusable when full auditing is turned on, how much auditing is enough? What are the alternatives? How does a particular technology fit with other technologies? Are there overlaps, gaps or conflicts? Is the technology right for the GCCS architecture? The most important question for me is "How does a technology affect the overall security of the system? Without this knowledge I find it difficult to make engineering tradeoffs when deciding the right mix of security technologies for the system. System level methodologies or frameworks to analyze security appear to be nonexistent.

4.1 State of the Art

Current security models don't seem to support the idea of the system level perspective of security. One of the first security models, the trusted computing base model from the government's Trusted Computer System Evaluation Criteria (Orange Book), was criticized for not addressing network issues and relying on the hardware and software within each workstation to enforce security policies. This model clearly lacks a system perspective. Network models have an implicit boundary that separates insiders from outsiders. Network models emphasize protective barriers that restrict outsiders from penetrating the system, however, there are many internal threats as well. Also, it may be difficult to determine the boundaries of the system in a network model.

The "How To" books and trade magazines of security often offer advice along the following lines:

1. Identify the system resources that need to be protected.
2. Identify the threats to the resources and/or system vulnerabilities.
3. Establish security policies.
4. Implement cost-effective strategies to minimize the risk threats impose against the resources.

Although the approaches vary slightly, they generally include these four steps. Although the books outline the approach, but they don't really provide any practical strategies. This last step is the kicker. As chief engineer, I found it relatively easy to identify system resources and threats for the GCCS. Implementing cost-effective strategies was difficult because I didn't have a way of comparing alternatives and it was difficult to understand how each alternative fit in the system context.

There has been extensive cryptanalysis research, attempts to discover stronger cryptographic algorithms, and theoretical research in intrusion detection. This type of research is invaluable if we are to rely on these technologies in our systems, but its place in the overall context must be understood. For example, encryption export controls present unique problems when the system must be compatible with foreign military systems. Trade magazines and security handbooks provide high level guidance on how to approach security, and some handbooks such as *Internet Security: Professional Reference* by New Riders Publishing provide very detailed information on how to build a firewall or how to set security sensitive system controls. Threat information taxonomies are easily found in most security textbooks and journals. The Computer Emergency Response Team (CERT) at the SEI periodically provides alerts and warnings about security problems and the Internet has a wealth of information about security. How does the system engineer pull the information together to see how all the policies, technologies and design maintain confidentiality, availability and integrity in a system?

5. A framework for security

As Chief Systems Engineer of GCCS, my integration tasks required that I see how each technology, design, or policy fit into the system. I wanted the framework to reveal the system security weaknesses and allow me to see how alternatives compared in the system. I felt such a framework would allow me to make cost-effective decisions about how to choose among all the things I could do to maintain a particular level of security within GCCS. I needed to be able to describe the level of the system security. Such a framework was not available to me at the time. I am now a Ph.D. student at Carnegie Mellon University and have the opportunity to work on constructing such a framework.

Instead of closing this experience with a wish list of questions for researchers to consider, I will lay out a preliminary sketch of the security framework that forms the basis of my own research. The framework takes advantage of the work accomplished by the Networked Systems Survivability Program and presented in *Survivable Network Systems: An Emerging Discipline* [ELL97]. The following outlines the components of a security analysis approach.

The System Security Analysis Framework (SSAF) is divided into five components: 1) the *system*, 2) *security technologies, policies and design* techniques, 3) known *weakness and flaws* for each item described in security technologies component, 4) *threats and vulnerabilities* and 5) the *security model*. SSAF provides a way to include both automated and non-automated security procedures as part of the analysis. The framework accommodates highly connected information systems and standalone systems. It is not constrained by the network topology, nor does it ignore the topology. The security model described in the framework places the system resources at the center of the model and provides a mechanism for showing how the system security mitigates the risk to those resources. The security model pulls all the other pieces together.

1) The *system* component of the framework describes the system architecture, relevant designs, and non-functional attributes. A complete system description that includes how people interact with the system is necessary so that the system engineer can understand how technologies, policies and designs are implemented or fit within the planned implementation. Many of the security technologies adversely impact the other non-functional attributes such as performance, so it is important to understand how the other non-functional attributes will be balanced in the systems. Non-functional requirements such as latency, reliability, and performance, must be identified here. The *system* component provides the context in which the security analysis takes place. Most of the information for the *system* component can be obtained from architectural description documents, design and requirement documents. Unfortunately, none of these documents were available for GCCS, however, most of the information could have been gathered from developers and software engineers.

2) The *technologies* component is a collection of security technologies, policies and designs that make up the system security. Security technologies include firewalls, access control lists, auditing mechanisms, intrusion detection systems, cryptography, etc. It is important to note that the technologies do not have to be part of the system. For example, if a system is a subsystem of a larger system and it relies on a firewall implemented by the larger system, then the firewall should be included in the list of security technologies. Security policies describe how system privileges are established, processes for reporting violations, password procedures, and any other policy that contributes to the overall security of the system. Configuration settings in products such as access control mechanisms or firewalls enforce many security policies; others are strictly procedural. Each element requires a detailed description about how it is implemented in the *system* described in system section.

3) The *weakness and flaws* component identifies known weaknesses and flaws of each of the items listed in the *technology* component. CERT disseminates bulletins to

alert system personnel of flaws in hardware and software products. Security policies often depend on the integrity of key individuals and systems suffer catastrophic failures when an individual betrays his trust. Separate analysis of weaknesses and flaws serves two purposes. First, analysis explicitly raises the awareness of the weaknesses and flaws associated with each item so that the system engineer can address these vulnerabilities, if possible. My experience has been that this is rarely done in practice. It was not done in GCCS. Second, it identifies areas that might need special attention when the system configuration changes.

4) The *threats and vulnerabilities* component addresses the system threats and vulnerabilities. Almost all security approaches advocate a threat identification step. None of the many threat assessment documents I have read provided specific guidance about threats and vulnerabilities. Documents usually identify a standard set of threats such as vulnerability to electronic eavesdropping, mal content employees, nuclear EMP, and hackers. Reports usually stated that hostile and non-hostile foreign countries might be highly interested in the information the system processed. Some reports might even identify a few flaws in the UNIX operating system for which there were known patches. These reports had relatively little value other than to confirm that I had followed the appropriate procedures and conducted a threat assessment. If I had not done so, the Joint Chiefs of Staff would have been prevented me from fielding the system. The *threat and vulnerabilities* component must be much more extensive if it is to be useful.

An initial start at improving threat assessments is a comprehensive taxonomy of threats. Fred Cohen [COHEN97] identifies 94 methods of attack. Additional detailed attack information is available from the Internet or from CERT bulletins. Security journal articles offer occasional guidance such as the recent article in Computer & Security [HAN98], which identified several attacks in detail. It may be impossible to collect all of the system threats because there are so many information sources and new attacks are appearing before the old attacks have countermeasures. Developing the *threat* component of the framework will probably be an ongoing process.

5) The core component of the framework is the security model (figure 2), which has four layers. The purpose of each of the other components is to help populate each of the four layers of the security model. System threats and vulnerabilities are external to the four layers. Each layer is populated with items from the technologies and policies component. The model is constructed using four defensive layers: 1) *protection*, 2) *detection*, 3) *mitigation*, and 4) *recovery*. Each layer plays a different role in protecting the system resources. Consistent

with other security models, the first step is to identify the system resources that must be protected. The *system* component should be the source of resource information.

The first layer is the *protection* layer. For each threat identified, the security engineer should identify the security technology or policy that stops the threat from gaining or denying access to a resource. This layer should be populated with all the security policies, products and designs that prevent an attack from succeeding. These policies and products have may have flaws, but they may still be effective against some (e.g. accidental) intrusions. Items that most likely fall into this layer are firewalls, passwords, background checks on employees, access control lists, etc. GCCS implemented all of these and more. Ideally, a system engineer would like a one for one mapping between threats and prevention mechanisms.

The second layer is the *detection* layer. Most likely, none of the mechanisms in the protection layer are 100% attack proof. There may not even be a protection mechanism for a particular threat. Hancock[HAN98] identified several attacks, some of which did not have known countermeasures. Without countermeasures, the system engineer needs to identify mechanisms that may detect an attack so that appropriate procedures are developed to properly react to an intrusion or denial of service attack. Intrusion detection systems, virus detection programs, audit trails and logs, special alerts and triggers are all security mechanisms that the security engineer should identify for the detection layer. System personnel should be guided by policy when responding to an attack. For each relevant threat, the system engineer should consider ways to detect an attack.

The third layer is the *mitigation* layer. Here the system engineer considers technologies and mechanisms that minimize the damage an attack may do if it is not detected or contained. System partitioning and system redundancy might be two techniques a system engineer could design into the system to minimize the damage from an attack. The purpose of this layer is to consider techniques and policies that help minimize the damage done from an intrusion that might go unnoticed for some time. Some of the attacks may not cause much damage because they are not particularly destructive attacks, so the system engineer may decide that a particular attack is more a nuisance that doesn't warrant any attention.

Recovery is the fourth layer. The system engineer must be able to recover from an attack. An attack may penetrate the preceding layers so the system engineer should consider how the system can recover from the damage. Back up and recovery procedures fall into this layer. Highly distributed systems like GCCS allow system engineers to design fail over and redundancy into the system without much trouble.

I have only begun to explore the feasibility and potential of this framework. Even if it does not immediately provide the quantitative analysis that most engineers hope for, I think it has potential to compare alternatives relative to one another. It pulls together the essential pieces of information in a uniform, structured way and gives the system engineer a system level perspective. If the today's security technologies can't prevent an attack then the next best answer is to detect and neutralize an attack. If the attack can't be detected then it is important to minimize the damage an attack can have on the system. Once the damage is done, the system must recover as quickly as possible.

If this framework had been available to GCCS it would have served us well. The information was available to populate the framework. The GCCS security checklist would have been an excellent starting place to gather an initial list to populate the *security technologies* component of the framework. Also, GCCS security specialists developed a GCCS security policy document that outlined many of the security policies that would be included in this part of the framework. Although these documents were available, there were many discrepancies between the policies identified in the document and those actually implemented. Obviously, it is important to distinguish between the written from the practiced.

Conclusion

GCCS presented many challenges. Security was the one area in which I felt the most helpless. It seems so much effort is put into each technology and so little effort into the engineering and design principles that need to guide system developers. Trade magazines don't provide the depth of advice that is needed to build the system security from the parts. The research community has not yet produced a model that is of direct, system-level assistance. If we don't understand how security integrates into system architectures today then how will know the role security plays in the domain architectures of the future?

References

[COHEN97] F. Cohen. Information System Attacks: A Preliminary Classification Scheme. *Computer & Security*. 16 (1997), p. 29-46

[ELL97] Ellison, R.J., Fisher, D., Linger, R.C., Lipson, H.F., Longstaff, T., Mead, N.R. Survivable Network Systems: An Emerging Discipline. Technical Report, CMU/SEI-97-TR-013, 1997

[HAN98] B. Hancock. Security Views. *Computer & Security* 17 (1998), p. 99-109

[I&RTS96] Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification (I&RTS), Version 3.0 (Draft) December 1996, Defense Information Systems Agency

[RUSS92] D. Russel and G.T. Gangemi, Sr. Computer Security Basics. Sebastopol, CA: O'Reilly & Associates (July 1992)

[WWMCCS92] *Modernization of the Worldwide Military Command and Control System*, National Academy Press, 1992