Exploitation of Security Vulnerabilities Inherent in Common Development of Process Control Systems

Freemon Johnson III
Intelligent Systems Division
National Institute of Standards and Technology
100 Bureau Dr. M/S 8230
Gaithersburg, MD 20899-8230
Freemon.Johnson@nist.gov

Abstract

According to a 2003 study commissioned by the Department of Commerce's National Institute of Standards and Technology, software bugs or errors are prevalent and so detrimental that they cost the U.S. economy an estimated $59.5 billion annually, or about 0.6 percent of the gross national product [18]. These errors translate into security vulnerabilities which are easily exploited. Understanding the inherent risk and consequences of these vulnerabilities is the subject of this paper. Specifically, security issues associated with client/server or producer/consumer software used in industrial control systems are addressed. As shown in the paper, the security issues differ over the system life cycle because security was not designed into the applications, but was added as an afterthought. For this reason, recommended changes to the development life cycle are proposed.

The dilemma and what to do about it

The definitive notion in this paper is network communications that utilize the Internet Protocol (IP) at the network layer and the Transport Control Protocol (TCP) and User Datagram Protocol (UDP) at their transport layers are at risk because of how Common Industrial Protocols (CIP) were designed to operate. This would encompass protocols such as Ethernet/IP, Modbus/TCP and others. This situation is further exacerbated by operators seeking remote access to process control environments over the Internet. Network premise security tools and equipment are “band-aids” to an Industrial Control System (ICS) network environment. The more severe the injury the more complex the remedy becomes which is analogous to a more complex and expensive solution. This paper shows that these layers upon layers of network security software and devices are not enough.

We begin with the Enterprise Information Security Architecture (EISA). EISA is the practice of applying a comprehensive and rigorous method for describing a current and/or future structure and behavior for an organization's security processes, information security systems, personnel and organizational sub-units, so that they align with the organization's core goals and strategic direction. Although often associated strictly with information security technology, it relates more broadly to the security practice of business optimization in that it addresses business security architecture, performance management and security process architecture as well.

EISA relies on the technology encompassed in the Open Systems Interconnect (OSI) layers 1 to 7 to be in place in addition to the security solutions for layers 8 through 15 for the Business Centric Model (BCM). In most cases, ICS rely on CIP families which depend on the protocol used, or any tailoring of that protocol to improve its performance. Even if you have secure methodologies to handle layers 8 to 15, the architecture will be susceptible to vulnerabilities because the foundation of the lower infrastructure is weak with respect to network security.

Even with all of the advancements in security technology and cryptology, there is always some inherent flaw that lies at the fundamental layers of any network. It is this lack of security in the protocol architecture which must be first addressed. Furthermore, as discussed in this paper, there is always a way to circumvent security tools and measures, because those tools are subject to the same weaknesses that govern their usage.

Recognizing this dilemma, the following is proposed: you cannot change every CIP stack within each router and operating system to prevent these vulnerabilities, but you can change the way your application communicates over them. To do so require “secure” code at the application layer and within operating system services level to mitigate the common risks regardless of the communication medium and protocols used.

Such an approach requires security to be designed into the software and system at every stage of its life cycle. The emphasis should not be placed on the “band-aids,” but more importantly on the development methodology of the software. The responsibility for security now lies with the developers of ICS client/server code, databases, data historians, the operating system services and so forth.

Security should be incorporated into the infancy stages of any system, especially software systems. This avenue is not pursued in the industry because of three reasons: first and foremost, it impacts time-to-market of the software and or device; second, the developers have to be retrained in secure coding practices which also impacts budgets and time-to-market because of re-training; and third, the need to support legacy and heritage systems of the existing infrastructure.

The first step to incorporate security that will persist throughout the software life cycle is to write secure code. National Institute of Standards and Technology (NIST) has a well defined process that is independent of the software language used, or the hardware description language residing in a field programmable gate array (FPGA) or application-specific integrated circuit (ASIC). In this paper, the NIST process is followed and discussed in the context of the vulnerabilities and risks associated with today’s ICS and traditional networking environments.

Major Conclusions

Before getting into the details, a summary of the major conclusions provides a roadmap of what to look for in the supporting rationale.

The System Development Life Cycle of any software application that plans to use the Internet to communicate should implement the methodologies defined in accordance to NIST standard Special Publication 800-64 REV. 1, July 13, 2004. The document is entitled “Security Considerations in the Information System Development Life Cycle” by Tim Grance, Joan Hash, and Marc Stevens. (23)

There are inherent risks associated at each level of the CIP stacks, OSI and BCM model with respect to security. Having the security tools and security management procedures in place today to mitigate risk is crucial.

At the most fundamental level, secure coding practices and integrating security into the development lifecycle at the infancy stages is critical for the ICS devices themselves. Of particular concern are those organizations offering services via the Internet. They need to take into account the security of their application from a developer perspective. Companies need to train their developers in this methodology with the same tenacity used to acquire their Capabilities Maturity Model Integration (CMMI) certification. Only in this case it would be with respect to using secure coding practices, techniques and methodologies throughout the product design and lifecycle.

Web Services Architecture Stack Layers 8-15 for BCM

To understand the weaknesses associated with the security for layers 8 to 15, one must first understand the architecture surrounding the implementation. The web architecture is different in comparison to the OSI model for layers 1 to 7. In particular, the web architecture layers are not distinct and autonomous from one another as in the OSI model. So let's first examine why.

The concept of security controls is important for understanding. The General Accounting Office defines controls to be the following (13):

- Access Controls: Restrict the ability of unknown or unauthorized users to view or use information, hosts, or networks.
- System Integrity: Ensures that a system and its data are not illicitly modified or corrupted by malicious code.
- Cryptography: Includes encryption of data during transmission and when stored on a system. Encryption is the process of transforming ordinary data into code form so that the information is accessible only to those who are authorized to have access.
- Audit and Monitoring: Help administrators to perform investigations during and after a cyber attack.
- Configuration management and assurance: Help administrators view and change the security settings on their hosts and networks, verify the correctness of security settings, and maintain operations in a secure fashion under conditions of duress.

How and where do these controls fit into various web architectures? Let's first observe the Web Services Architecture Stack shown in Figure A:

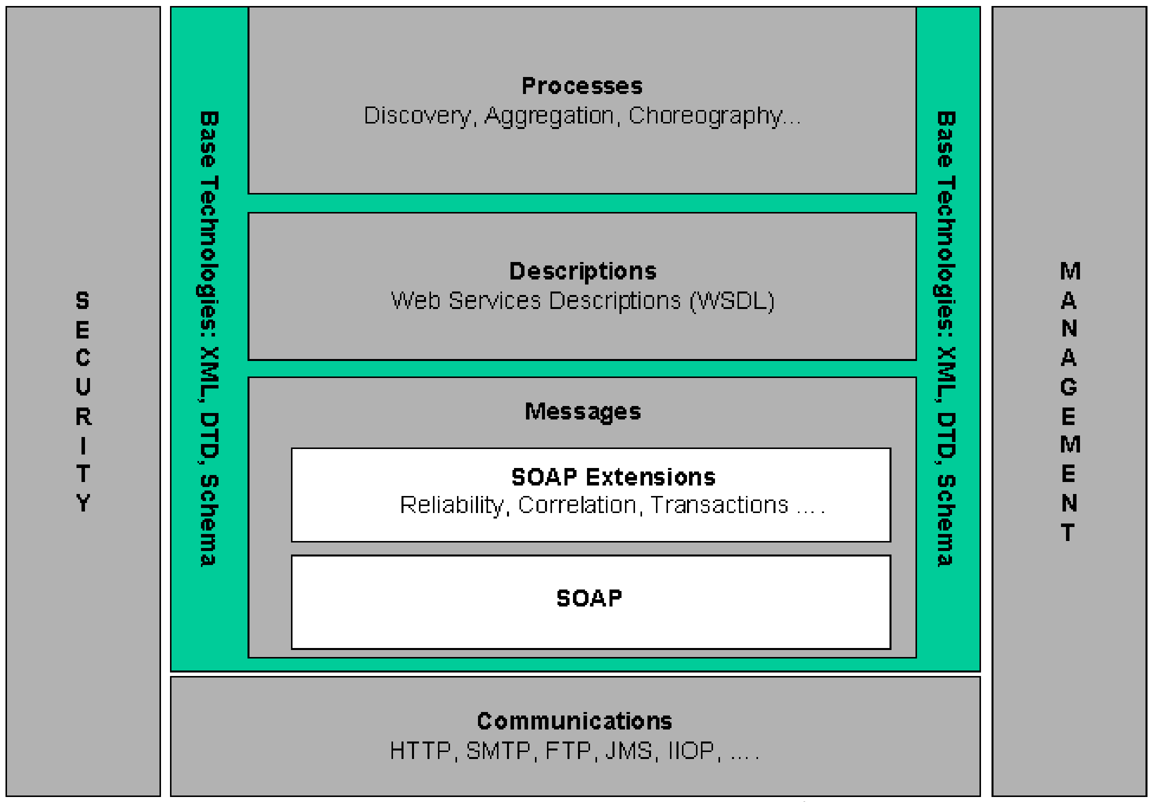
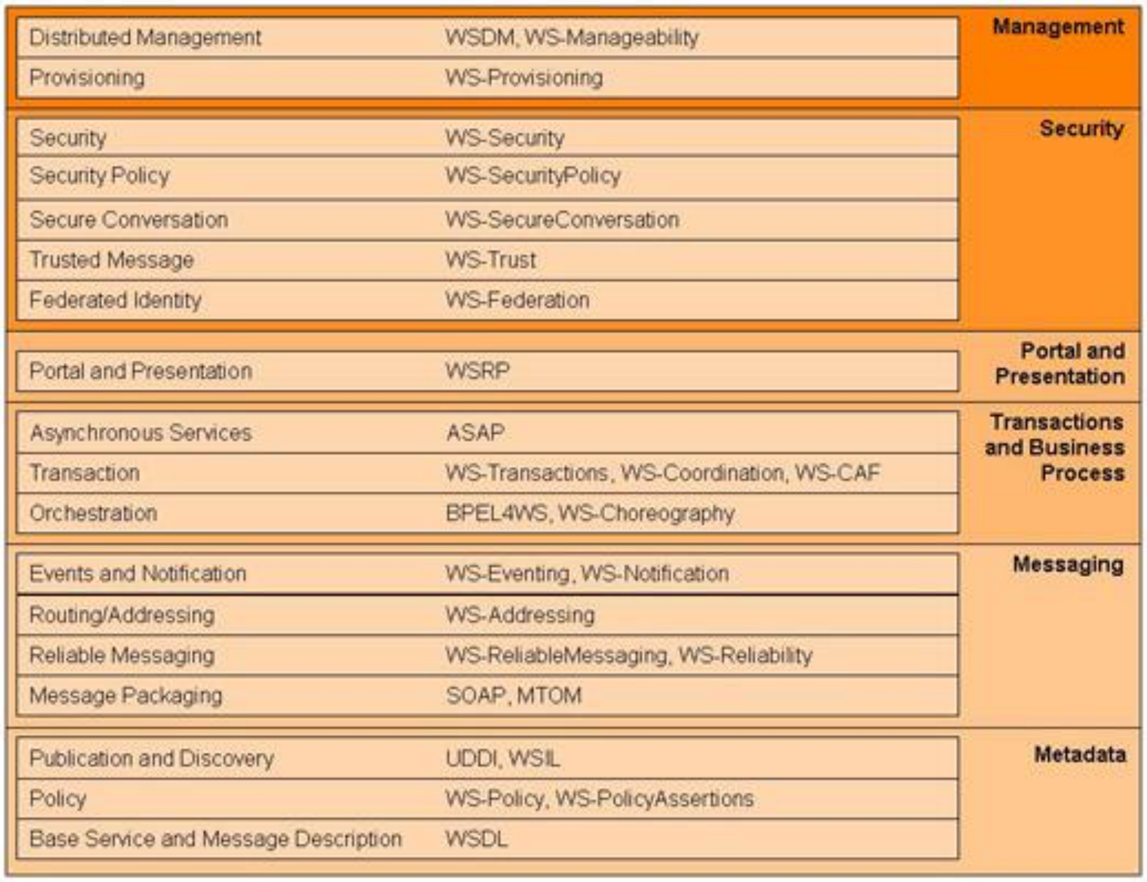

SECURITY
Base Technologies: XML, DTD, Schema
Processes
Discovery, Aggregation, Choreography...
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions ....
SOAP
Base Technologies: XML, DTD, Schema
MANAGEMENT
Communications
HTTP, SMTP, FTP, JMS, IIOP, ...

Figure A. Web Service Architecture[1]

1 *http://www.w3.org/TR/2003/WD-ws-arch-20030514/*
Security is parallel to management and leads the reader to believe that security envelops all aspects of the architecture. However, another perspective is shown in Figure B. Web Services Stack which shows the following:



Distributed Management	WSDM, WS-Manageability	**Management**
Provisioning	WS-Provisioning	
Security	WS-Security	**Security**
Security Policy	WS-SecurityPolicy	
Secure Conversation	WS-SecureConversation	
Trusted Message	WS-Trust	
Federated Identity	WS-Federation	
Portal and Presentation	WSRP	**Portal and Presentation**
Asynchronous Services	ASAP	**Transactions and Business Process**
Transaction	WS-Transactions, WS-Coordination, WS-CAF	
Orchestration	BPEL4WS, WS-Choreography	
Events and Notification	WS-Eventing, WS-Notification	**Messaging**
Routing/Addressing	WS-Addressing	
Reliable Messaging	WS-ReliableMessaging, WS-Reliability	
Message Packaging	SOAP, MTOM	
Publication and Discovery	UDDI, WSIL	**Metadata**
Policy	WS-Policy, WS-PolicyAssertions	
Base Service and Message Description	WSDL	

Figure B. Web Services Stack[2]

Layers 8 and 9 of the Web Services Architecture Stack

The goal of web services (WS) security as defined in their specification is to enable applications to construct secure Simple Object Access Protocol (SOAP) message exchanges. This specification provides a flexible set of mechanisms that can be used to construct a range of security protocols; in other words this specification intentionally does not explicitly define a single security protocol.

As with every security protocol, significant efforts must be undertaken to ensure that the protocol constructed is not vulnerable to a wide range of attacks. To this end, the focus of this specification is to describe a single-message security language. It provides message security for an established session, security context and/or policy agreement.

Network Security Solutions for OSI Layers 1-7

Several mechanisms must be considered to secure OSI layers 1-7. Each of the following mechanisms contributes to a Defense-in-Depth (DiD) security strategy (9).

2 *http://roadmap.cbdiforum.com/reports/protocols/*

Routers/Switches:

There is significant research being conducted in router vulnerabilities, especially in terms of Denial of Service (DoS) attacks. A current study at Massachusetts Institute of Technology is investigating *Mayday* architecture, which consists of a lightweight authenticator, filter ring, and overlay nodes that distinguish the difference from trusted traffic versus attack traffic. This study is one of many that are attempting to thwart DoS attacks. The architecture case studies focused on eavesdropping and various forms of attack (1). The studies are limited and still allude to risk, which strengthens our vision that software must be security-smart.

Firewall Technologies:

Firewalls, whether stateful or not, are subject to following deficiencies:

- Physical access to facilities, which house this equipment, must be secure.
- If a Virtual Private Network (VPN) is configured with encryption to enter a firewall, and that connection is compromised, the user provides a secure tunnel for a hacker to enter the network.
- Firewalls easily become single points of failure, which make them an easy target if that is the only way into the network.
- If a firewall can be configured remotely, a hacker has the ability to reconfigure the firewall remotely as well.
- Log information that resides with a firewall is subject to any compromises of the firewall itself.
- Firewalls do not protect against traffic profiling. Therefore, enumeration of the network surrounding the firewall can be studied. Egress traffic can be captured. Logging that is not executed behind the firewall can be discovered. Domain Name Servers (DNS) information, proxies, etc. that are in Demilitarized Zones (DMZ) can be discovered as well.
- Firewalls are useless to configurations dealing with Dynamic Host Control Protocol (DHCP). An entire subnet cannot be trusted, because all it takes is one host within the subnet to be compromised for DHCP to become vulnerable.
- "95 % of attacks occurred despite a firewall was in place" (7).

Intrusion Detection Systems (IDS):

Intrusion Detection and or Prevention Systems are subject to the following conditions:

- An IDS is only as good as the administrator configuring the Access Control List (ACL) rules to search for suspicious behavior. Secondly, the administrator parsing the logs is also challenged by information overload. It is difficult to parse through a multi-megabyte or more of logs, which makes it easy to overlook a crucial piece of evidence that may lead to a perpetrator. Because the logs are so large, storage becomes an important factor in cataloging the network events that transpire during the course of day.
- In addition to dynamic logging, correlation tools are needed to compile the pieces of the infraction/anomaly to complete the investigation. IDS tools do not usually comprise these tools. Parsing voluminous logs is not trivial. On high-speed networks, the traffic log can increase exponentially in size if the rules are too general. If the rules are too specific, it leaves room for neglecting an important factor that could be exploited and is never logged.

- Another limitation of the IDS is the response time to an event. Even if an attack is noticed, a solution is not provided that attempts to resolve the issue once it is in progress (6). It has been researched that "61 % of attacks occurred despite an IDS was in place" (7).

WI-FI/Bluetooth Technologies:
Wireless technologies are susceptible to vulnerabilities that are different from wired networks in the following ways:

- The security vulnerabilities of Wi-Fi base stations are varied. Anyone with an enhanced transceiver or antennae can pick up a Wi-Fi network without having to be within local proximity of the base station. Network traffic can be captured, sniffed, and the base station can serve as the gateway to the offender. One of the traditional standard encryption algorithms utilized over the radio frequency (RF) link is Wired Equivalency Protection (WEP). UC Berkeley cracked WEP's 40-bit algorithm in less than four hours using 250 workstations. They proved that no matter what the size of the key, WEP is susceptible to attacks. They cracked both the 40 bit and the 128 bit versions equally well (12).

- One of the chief concerns for businesses about the Wi-Fi wireless networking technology has been the lack of a security standard. The Institute of Electrical and Electronics Engineers has been working to develop and approve 802.11i or WPA2, a security standard that won't be finished for at least another year. The latest security specification, Wi-Fi Protected Access (WPA), is a subset of what will become the 802.11i standard. WPA replaces the existing security protocol, called Wired Equivalent Privacy.

- WPA is the third specification related to Wi-Fi to receive certification from the Wi-Fi Alliance for interoperability, which means that approved products are supposed to work with each other no matter which company manufactured the product. The certification is meant to broaden Wi-Fi's reach and expand the number of networks that people will be able to access. The Wi-Fi Alliance has approved products based on the 802.11a and 802.11b, and 802.11g-based products.

Cryptology:

Despite the advances in cryptography, cryptanalysts will not argue against the following points of concern:

- As with other devices, the area of cryptology has security concerns as well. The key space has grown logarithmically too quadratically in the last several years because systems are becoming more and more powerful thanks to Moore's law: cheaper and faster memory and faster bus speeds of computers abroad. This produces derivatives such as more CPU utilization, and more overhead in bandwidth because of the key size. The key will have to expire at some point as well. Algorithms that protect your data today are cracked tomorrow, figuratively speaking. Users must constantly keep re-encrypting data to keep up with technology and protect themselves against hackers who also keep up with technology (5). Traditional algorithms are too dependent upon the randomness, or entropy of the cryptographic variables that are used as keys for input variables. Studies are being conducted to create "true" random number generators in Field Programmable Gate Arrays (FPGAs) and Digital Signal Processors (DSP), because white noise is an excellent random source (13, 14). To go one step further, the day that quantum computers become a reality years from now, all forms of encryption will be susceptible to being compromised regardless of the key size, algorithm, or randomness of the Pseudo-Random-Number-Generator (PSNG).

- Algorithms today use Cipher Block Chaining (CBC) or even Accumulated Block Chaining (ABC). Both of these encryption schemes are susceptible to various forms of attacks. They both prove to be bandwidth consumers, because CBC is not length preserving. CBC is not a cipher either but an encryption scheme because it changes with each block of data that is encrypted. The blocks are not fixed. If the blocks were fixed, the randomness that is used in the equation to encrypt the data is no longer random. The author of Online Ciphers and Hash-CBC Construction of the International Association of Cryptologic Research (IACR) states that you need a Pseudorandom Permutation (PRP) which is a secrete or an oracle that is not computationally feasible to be used as a key at the initialization of most CBC encryption algorithms. It again goes back to pseudo-random number generators randomness. The author proposes this method for the benefit of online cipher to enhance performance without the need for buffering because the packets are too large. The constraints needed to make this a reality are still difficult to satisfy (10).

- With regards to Transport Layer Security (TLS)/ Secure Socket Layer (SSL), Internet Protocol Security (IPSEC), and Secure Shell (SSH), IPSEC is proven to be the most secure. All of the methodologies use symmetric authentication Message Authentication Code (MAC) which are different in the following ways: Secure Socket Layer (SSL) authenticates the packet then encrypts the data; IPSEC encrypts data first and then authenticates the packet, and SSH encrypts and authenticates the packet at the same time (15). SSL has a problem with guaranteeing authentication (non-repudiation), but not with the secrecy (confidentiality) of the packet sent.

- Lastly, in today’s times, the legal implications of cryptographic algorithms and technologies are significant. One cannot create or utilize technologies that the federal government does not know about and can control via import/export laws as a matter of National Security. An example of the ramifications of disobeying the federal government can be referenced from Phil Zimmerman’s grand jury indictments for the creation of Pretty Good Protection (PGP) (16). Cryptology also comes at price because the algorithms and applications are patented in most cases. Security Dynamic’s, the creator of proprietary smart cards, has been receiving royalties for all products using RSA encryption algorithms, regardless if whether their algorithms were even used for pre 2000 systems It was not until September 2000 when the patent expired (2).

CIP Protocol Stack for Layers 1 – 7

The CIP family, consisting of Ethernet/IP (Industrial Protocol), CompoNet, DeviceNet, and ControlNet, is a protocol stack developed for the purposes of deterministic, simple, power efficient, ways for devices within an ICS to communicate as well as disseminate data.(20) These protocols were never designed with security as a requirement in their infancy. One could argue that since most process control facilities have closed networks, the need for remote capabilities was minimal and usually consisted of landline for direct dial-in to a control systems if needed by operators. Since most Programmable Logic Controllers (PLCs) and similar devices do not have the processing capability beyond simple instructions, incorporating traditional security components such as authentication, confidentiality, integrity, etc. is not trivial or even practical in most cases within a CIP network.

The CIP family beginning with DeviceNet was not created until 1994. The CIPSafety extensions were not introduced to the protocol until October of 2002. CIPSafety resides at the application layer. CIPSafety can be applied to the CIP family and provide what is known as a Safety Integrity Level (SIL) IEC61508. CIPSafety defines a topology that supposedly guarantees the following attributes: tolerance to high noise, automatic checking of duplicate node addresses, re-tries at the data link layer, priority configuration, low Bit Error Rate (BER) of 10^{-7} or better, error counters, and connection-based messaging as a way of reporting faults.

As stated previously, the extensions reside at the application layer which is even higher than traditional TCP/IP security efforts that start at the network layer through the application layer. Because having homogenous hardware is critical for characterization, capabilities, and determinism, customized hardware with safety features is not a consideration within the CIPSafety extension topology. The extensions are objects defined solely in the application layer of the device that is communicating on the ICS network.

With that said, the producer/consumer model is much like publish/subscribe in other networks. Most of the communication is simple with regards to either point-to-point (explicit messaging) or multicast architectures (implicit messaging). Opening a port or a forward open connection is analogous to opening a socket. Since Ethernet/IP uses traditional TCP and UDP for its transport and just encapsulates the CIP application payload, it to susceptible to the same vulnerabilities as traditional TCP/IP stacks. DeviceNet, ControlNet, and CompoNet have simple communication procedures where security was never forethought as well, and also have an open/listen simple methodology of communication. This leaves the notion that one can either enhance the protocol stack for secure code practices as well as the application itself that uses the stack to communicate that resides in the device or tool.(19)

Secure Code Practices/Techniques

It has been reported that the "Microsoft Security Response Center estimates the cost of issuing ***one*** security bulletin and the associated patch cost $100 000, and that's just the start of it" (8). Clearly, there is a business case for secure code practices and well managed software development to ensure effective security is implemented and adequately tested throughout the software development life cycle (SDLC).

As stated earlier, the initial training of the developers to write secure software is paramount. Although this process usually increases cost during initial phases and time, the investment is invaluable to a company's marketability

Buffer overflows account for more than 50 % of all CERT security advisories, year after year (14). Choosing the right programming language can alleviate most of these types of problems. Languages that are recommended are: JAVA, C# and Python rather than ANSI C/C++. If the developer uses C language, he or she must be aware of poorly written C libraries that have no bounds checking on array and pointer references (21). For example strncat(), although better than strcat(), allows the developer to set a limit on the string concatenated to an existing buffer. (11) However, it does not keep track of the usage of the entire buffer as strings are repetitively concatenated. The developer has to keep track of the usage of the buffer.

Other Layers of Network Security (Social Engineering)

The Defense Information Systems Agency (DISA) reports that "96 % percent of successful attacks could have been prevented if users had followed protocols" (17). Every layer of system security has associated costs and additional steps for the users. The cost and inconvenience (to the users) associated with security must be balanced against the cost and inconvenience of the loss to availability, integrity, or confidentiality (18).

The System vs. Software Development Life Cycle

The distribution of services in a producer/consumer or client/server environment for ICS increases the susceptibility of damage from viruses, fraud, physical damage and misuse. As noted in the previous section, network technologies collectively provide some security. However, as organizations move towards multi-vendor systems, remote administration, use of extranets, often chosen on the basis of cost alone, the security issues multiply. In addition, a single vulnerability or single point of failure in an interconnected system results in risk to the entire network and its components. As a result, the entire interconnected system should be evaluated for the impact to its security.

Boundaries and the Client/Server or Producer/Consumer Environment:

First, one must define a system. According to Federal directives, boundaries of a "system" are defined by four elements: "(1) must be under the same direct management control; (2) must have the same function or mission objective; (3) have essentially the same operating characteristics and security needs; and (4) must reside in the same general operating environment" (19). An example of a "system" with these four elements would be an organization's financial management system. Note that a system in this context is not solely software. Rather, it may be numerous applications, interconnected with numerous servers or computers, or other assets. It is imperative to inventory, or have knowledge of, all assets or components of a single system.

Client/server computing comprises three building blocks:

- ❑ Client/Producer
- ❑ Server/Consumer (may be more than one)
- ❑ Network (allowing the client/server to communicate).

A logical and physical separation exists between the client and server and the client/server system coordinates the work of both of these components and efficiently uses each one's available resources to complete assigned tasks. This separation of client and server provides an open and flexible environment where mix and match of hardware and operating systems is the rule. The network ties everything together. Today the client applications run predominantly on Human Machine Interface (HMI) or desktop computers connected to a network. The servers are also connected to the network and know how to connect to their clients.

The distributed nature of client/server applications prompts the need to manage integration with other components. This management can be facilitated by a methodology, which reinforces an engineering discipline, not only in software development, but in interconnection and operation with other devices and software. The methodology is known as the "System Development Life Cycle (SDLC)." The software development life cycle is a component of the SDLC, sometimes overlapping, while complementing one another at other times. The phases of software development demand a systematic, sequential approach, which typically deals with the requirements of a particular product or application. Figure C. shows the relationship between the typical “software” phases, and the “system” phases.

SOFTWARE DEVELOPMENT LIFE CYCLE	**SYSTEM DEVELOPMENT LIFE CYCLE**
Requirements	Initiation
Specification	Development/Acquisition
Design	
Implementation	Implementation
Integration	Operation
Delivery	
Maintenance	
Retirement	Disposal

Figure C. Comparison of System and Software Life Cycle Phases

The point of this paper is that despite which methodology is used, both the system and software life cycle must include security considerations. As discussed earlier, often the problems associated with applications, daemons, services, and hardware results from a poor manufacturing process with respect to security. This can have an impact every component or asset contained in a system. NIST has developed 33 engineering principles for securing information systems. The principles provide a structured approach to which information security can be considered. Each phase of the system development life cycle will be reviewed, there collectively 33 security principles associated but not necessarily inclusive with each phase as defined by NIST (22).

Principle 1: Establish a sound security policy as the "foundation" for design.
Principle 2: Treat security as an integral part of the overall system design.
Principle 3: Clearly delineate the physical and logical security boundaries governed by associated security policies.
Principle 4: Reduce risk to an acceptable level.
Principle 5: Assume that external systems are insecure.
Principle 6: Identify potential trade-offs between reducing risk and increased costs and decrease in other aspects of operational effectiveness.
Principle 7: Implement layered security (Ensure no single point of vulnerability).
Principle 8: Implement tailored system security measures to meet organizational security goals.
Principle 9: Strive for simplicity.
Principle 10: Design and operate an information system to limit vulnerability and to be resilient in response.
Principle 11: Minimize the system elements to be trusted.
Principle 12: Implement security through combination of measures distributed physically and logically.
Principle 13: Provide assurance that the system is, and continues to be, resilient in the

face of expected threats.
Principle 14: Limit or contain vulnerabilities
Principle 15: Formulate security measures to address multiple overlapping information domains.
Principle 16: Isolate public access systems from mission critical resources (e.g., data, processes, etc.).
Principle 17: Use boundary mechanisms to detect unauthorized use and to support incident investigations
Principle 18: Where possible, base security on open standards for portability and interoperability.
Principle 19: Use common language in developing security requirements.
Principle 20: Design and implement audit mechanisms to detect unauthorized use and to support incident investigations.
Principle 21: Design security to allow for regular adoption of new technology, including a secure and logical technology upgrade process.
Principle 22: Authenticate users and processes to ensure appropriate access control decisions both within and across domains.
Principle 23: Use unique identities to ensure accountability.
Principle 24: Implement least privilege.
Principle 25: Do not implement unnecessary security mechanisms.
Principle 26: Protect information while being processed, in transit, and in storage.
Principle 27: Strive for operational ease of use.
Principle 28: Develop and exercise contingency or disaster recovery procedures to ensure appropriate availability.
Principle 29: Consider custom products to achieve adequate security.
Principle 30: Ensure proper security in the shutdown or disposal of a system.
Principle 31: Protect against all likely classes of "attacks."
Principle 32: Identify and prevent common errors and vulnerabilities.
Principle 33: Ensure that developers are trained in how to develop secure software.

Given these principles, we now address the questions that need to be answered and what issues need to be addressed in each phase of the life cycle.

Initiation Phase

During the initiation phase, a sensitivity assessment should be conducted which evaluates the criticality of the information to be processed and how the computers will process or host this information. For example, the following questions should be answered in this phase:

- What am I trying to protect?
- How does the information relate or contribute to the organization's overall mission?
- What am I trying to protect against?
- What if the information were disclosed (confidentiality)?
- What if the information were modified (integrity)?
- What if the information were not available (availability)?

Development/Acquisition Phase

During the development/acquisition phase, security requirements are created and incorporated into design specifications along with assurances that security features acquired can and do work correctly and effectively. Security requirements may include technical features (e.g., access controls), assurances (e.g., background checks for developers), operational practices (e.g., awareness and training, contingency plan, system security plan, user manual, operations/administrative manuals), test plans/script/scenarios, and costs associated with background checks.

As noted, specifications should include security requirements. For systems that are being built, security features need to be developed, the development process monitored for security problems, changes responded to, and threats monitored. If commercial off-the-shelf products or systems are used, the product or system should be monitored to ensure security is part of market surveys.

Solicitation documents should be contracted, and the product or system evaluated before it is widely deployed.

Implementation Phase

The implementation phase includes all activities in support of configuring, enabling, testing, and verifying the system. The risk management process supports the assessment of system implementation against its requirements and within its modeled operational environment. Decisions regarding risks identified must be made prior to system operation. (22)

The development of client/server applications has several risks. The first risk is the skill level of the client/server/producer/consumer development team. In dealing with new products, the network, and a new operating environment, the development team may not be fully experienced. To compensate for this risk, it is imperative that a management policy be written that requires a peer review of all new client/server application designs. This review would be performed by an internal or external expert as a matter of policy. From the review procedure a better overall design could be accomplished and cross training of experience could be transferred.
Second, there are risks inherent to the design methodology. In the client/server world the application development process takes on a Rapid Application Development (RAD) approach. With this approach, the application development is designed for an expeditious deployment. Because of this aspect there may be a tendency not to use a formalized structured development methodology. This rapid approach may serve as a quick solution but may not provide the application to the interoperability that will be necessary to quickly modify the application to take advantage of new hardware, software, or corporate goals.

To offset these risks without restricting the application development process, it would be wise to establish a data classification methodology that would help to define organizational data into four classes. The highest class would be organizational data. The next class would be divisional data, then group, and finally user or local data. Using this classification, an organization could employ a quick risk assessment for any application that uses organizational, divisional, or group data. The level of risk would be mapped into the number of steps required to meet the minimum methodology standard. The higher the risk the more steps required. In addition, this classification methodology could be used to store application objects in a repository at each level. This would allow for their reuse for other application development processes. Finally, the classification methodology could be used to tag the data and track its movement throughout the network. With this approach corporate standards, procedures, and documentation requirements all become part of the application development process.

The third risk is repository control over objects for the client/server application. These objects are typically represented in both source and object form. They include menus, programs, scripts, and windows. Only the version of these objects should be stored within the user environment. Using a version control or check-out/check-in control over the updating of objects will maintain integrity and control over the application's execution. The original source code should be placed on a protected library, backed up, and stored at an off-site location for additional protection. Besides version control, the system could also be set up to verify the integrity of critical objects by using a check sum total on these objects when the workstation signs on to the file server.

Test data should be developed and applied to a small unit, subsystem, and then the entire system. The system's security features should be configured and enabled. Testing should be done in a systematic manner. The following list specifies sample tests one may include at the input level, and those at the host or computer level, as follows. This information was extracted from James Whittaker's book on "How to Break Software." (24)

- ✓ Apply inputs that force all the error messages to occur.
- ✓ Apply inputs that force the software to establish default values.
- ✓ Explore allowable character sets and data types.
- ✓ Overflow input buffers.
- ✓ Find inputs that may interact and test combinations of their values.
- ✓ Repeat the same input or series of inputs numerous times.
- ✓ Force different outputs to be generated for each input.
- ✓ Force invalid outputs to be generated.
- ✓ Force properties of an output to change.
- ✓ Force the screen to refresh.
- ✓ Apply inputs using a variety of initial conditions.
- ✓ Force a data structure to store too many or too few values.
- ✓ Investigate alternate ways to modify internal data constraints.
- ✓ Experiment with invalid operand and operator combinations.
- ✓ Force a function to call itself recursively.
- ✓ Force computation results to be too large or too small.
- ✓ Find features that share data or interact poorly.
- ✓ Fill the file system to capacity.
- ✓ Force the media to be busy or unavailable.
- ✓ Damage the media.
- ✓ Assign an invalid file name.
- ✓ Vary file access permissions.
- ✓ Vary or corrupt file contents.

Security and sensitivity assessments documented in the previous phase should be reviewed. These requirements include review of the security management (administrative controls, safeguards), physical facilities, personnel, responsibilities, job functions, and interfaces, procedures (e.g., backup, labeling), contingency planning, and disaster recovery plans.

Operation/Maintenance Phase

During the operation/maintenance phase, the system performs its work. The system is almost always being continuously modified by the addition of hardware and software and by numerous other events. Activities during this phase include, but are not limited to, security operations and administration, operational assurance, and audits and monitoring. Examples include: performing backups, holding training classes, maintaining user administration and access privileges, ensuring audit logs are available, patching software, reviewing physical protection, reviewing off-site storage usage, services, and availability, reviewing input and output distribution processes, reviewing warranties, reviewing the actions of people who administer the system and those who use the system (e.g., change control procedures), reviewing technical controls, reviewing interdependencies, and comparing documentation to the state of the system. During this phase, one should also perform periodic self-administered or independent security audits (risk assessments). Risk assessments may include using automated tools, an internal control audit, security checklists, or penetration testing.

Audit types can provide information about the management, operation, and technical implementation of security through a self-audit or an independent audit because it is typically conducted in “real-time”. Monitoring is susceptible to legal review, and may include review of system logs, use of automated tools, configuration management, monitor of external sources of information, and formal review of the system and processes. (22)

Disposal Phase

The disposal phase involves the sanitization of some or all components of the system, including the information, hardware, and software.   Activities include moving, archiving, discarding, or destroying information and clearing the media. In some cases, the disposal phase may include “debriefing” users.

For encrypted data, ensure long-term storage of cryptographic keys. Consider legal requirements for records retention or consult with organization policy regarding retaining and archiving records. Depending on the life of the information, and the nature of its sensitivity (as defined in the initiation phase), overwrite, degauss, or destroy the information.

Conclusion

By following the System/Software Development Life Cycle and involving the engineering principles in its design from infancy, a developer can mitigate the risk of their software product being compromised from malicious or non-malicious intent. The benefit of incorporating security in the early stages of development prevents loss of credibility, conserves specialized personnel, and minimizes compatibility, and integration issues. These factors can all be reduced to their lowest terms, which are most importantly, time and cost. Once company’s marketing executives understand this, they will realize that these benefits surely outweigh “time to market” and by doing so will build longer lasting relationships and confidence of their market’s consumers. Software products will maintain their integrity during their life cycles over the course of iterations and enhancements, until retirement.

References

1.) Andersen, David G. 4th USENIX Symposium on Internet Technologies and Systems. Mayday: Distributed Filtering for Internet Services. Seattle, 2003.

2.) Brenton, Chris. SANS Institute. VPNs and Remote Access. Baltimore, 2001.

3.) Brenton, Chris. SANS Institute. Firewalls 101: Perimeter Protection with Firewalls. Baltimore, 2001.

4.) Brenton, Chris. SANS Institute. Firewalls 102: Perimeter Protections and Defense, In-Depth. Baltimore, 2001.

5.) Brute Force Attacks on Cryptographic Keys. University of Cambridge, Computer Science Department. August 3, 2001. <http://www.cl.cam.ac.uk/~rnc1/brute.html>

6.) Cox, Phillip C. and Mark K. Mellis. K3Media Group Networld + Interop. Intrusion Detection and Network Forensics. Las Vegas, 2002.

7.) CSI/FBI Computer Crime and Security Survey, Computer Security Institute. April 7. 2002<http://www.gocsi.com>.

8.) Dewire, D. T. Client/Server Computing. Singapore: McGraw-Hill, 1993.

9.) Engineering Principles for Information Technology Security. NIST Special Publication 800-27. Washington: Government Printing Office, 2002.

10.) Generally Accepted Principles and Practices for Securing Information Technology Systems. NIST Special Publication 800-14. Washington: Government Printing Office.

11.) Howard, Michael and David LeBlanc. Writing Secure Code. Washington: Microsoft Press, 2003.

12.) Kilian, Joe., ed. Cryptographic Hardware and Embedded Systems. International Association of Cryptologic Research. Random number generators founded on signal and information theory. Worcester, 1999.

13.) Information Security Technologies to Secure Federal Systems **GAO-04-467**, The Government Accounting Office, 2004.

14.) Layers One &Two of 802.11 WLAN Security**.** SANS (SysAdmin, Audit, Network, Security) Institute. August 3,2001. <http://www.sans.org/rr/wireless/WLAN_sec.php>

15.) McClure, Stuart, Saumil Shah and Shreeraj Shah. Web Hacking Attacks and Defense. New York: Addison Wesley, 2003.

16.) McGraw, Gary and John Viega. Building Secure Software. New York: Addison-Wesley, 2002.

17.) Message Authentication. University of California at Davis, Computer Science Department. <http://wwwcsif.cs.ucdavis.edu/~rogaway/papers/research.html#ma-res>

18.) National Aeronautic and Space Administration Goddard Space Flight Center. IP-in-Space Security Handbook. Publication S-38657-GTOR16. Greenbelt: 2001.

19.) ODVA Safety Networks, Increase Productivity, Reduce Work-Related Accidents and Save Money. White Paper.

20**.)** ODVA The Common Industrial Protocol (CIP) and the Family of CIP Networks, Ann Arbor, MI, 2006.

21.) OMB Circular A-130. Office of Management and Budget. Washington.

22.) Risk Management Guide for Information Technology Systems. NIST Special

Publication 800-30. Washington: Government Printing Office, 2001.
23.) Security Considerations in the Information System Development Life Cycle, NIST Special Publication 800-64 REV. 1, Washington: Government Printing Office,July 13, 2004. .
24.) Whittaker, James A. How to Break Software: A Practical Guide to Testing.