Application Systems Development

Lowell Bruce McCulley
IT Security Professional, Troy, New Hampshire, U.S.A.

Abstract
This entry surveys some information security considerations pertinent to application systems development, reviews a number of areas related to application systems and the technical and organizational development environments, and describes a novel tool for incorporating security engineering into the application development process.

If carpenters built houses the way programmers build programs, the first woodpecker that came along would destroy civilization.
—Weinberg's Second Law of Computer Programming

Woodpeckers are just attempting to remove bugs.
—Further commentary by Weinberg

Jerry Weinberg was actually commenting on the state of the art in software engineering in the 1960s, not present-day security engineering, when he authored his second law. The fact that his comment is as pertinent to today's malicious hackers as it was to innocent practitioners of by-gone days illustrates the fundamental truth that security is an inherent attribute of well-designed information systems. His additional commentary points out that systems-engineering activities (e.g., debugging) destabilize systems, clashing with the security imperative for stable systems. This entry suggests that enlisting woodpeckers (or systems developers) in the security effort benefits both security and development. We posit that it is best to justify information security programs on economic issues in the management hierarchy by showing value from cooperating on technical issues in the project arena. The best way to benefit the development team and the entire organization is by working in harmony with development priorities, so we present several ways to do so.

We begin by surveying the existing state of the art in information security programs, in which we identify some things that do not work as well as they might. Economic factors are discussed as the fundamental drivers of management decisions about technology, applications systems, and security. We proceed to an examination of the nature of application systems and associated technologies, to better define our focus and the scope and bounds of our concerns. This leads to a review of the systems' development life cycle that applications follow, to understand how the development activities and security concerns change at different stages in the existence of applications systems. Finally, we introduce an innovative approach to using a new security engineering tool in a way that generates value for the systems development process. We close by discussing the integration of that approach into the systems development life cycle, and identifying some potential directions for subsequent research and development.

STATE OF THE ART IN BUSINESS APPLICATIONS SYSTEMS SECURITY

A paradigm shift seems needed in our approach to securing business information systems.

The fundamental shift is to position security as a value enhancer throughout the application systems life cycle, especially the development engineering process. Application systems security would benefit from several effects of this shift, based on decades of experience developing critical systems. The reason is that business organizations often resist rather than promote security programs, on economic grounds. Application systems are the most important point of focus, because they are the raison d'etre for information systems (and thus for information security) in the business world. To successfully accomplish this, we must first understand several things, including economic factors, the nature of application systems and their life cycle, security drivers, and even historical context. This entry presents a framework and some tools to help integrate security into the application systems development process as a value enhancer.

Dr. Peter Tippett, CTO of TruSecure, wrote:

For years, the focus of most security efforts has been centered on identifying and then fixing vulnerabilities in technology. The prevailing belief is that if a hole is found in the IT armor of an organization, it should be fixed immediately before it can be exploited by some cyber-deviant. While this approach sounds logical and effective, it is actually the beginning of a vicious cycle that occupies...
vast amounts of time and wastes several millions of corporate, government, and consumer dollars every year.\(^1\)

Dr. Tippett goes on to draw an analogy with healthcare, saying:

The current approach to security would also have us inoculated for the most minor of illnesses, and protected against every possible cut, bruise, or blister… which is both ineffective and impractical. Medicine has progressed beyond this piecemeal approach by taking a holistic view of the organism and by emphasizing prevention as the best cure. Unfortunately information security has not followed that model, at least not yet, but it suggests a framework to use as a model to improve our struggling InfoSec efforts. We need to extend our focus to view information systems as functional entities rather than collections of technical components, and to define and address security concerns in that holistic context. By doing so, we also have the opportunity to transform our security efforts from a costly burden into a valuable benefit.

Securing Web-based business-to-business E-commerce application systems poses new problems requiring a new approach to engineering security into the application systems development life cycle. A typical Web-based application utilizes external (e.g., Internet) connections from existing segmented network infrastructures that provide a layered defense-in-depth. The external connections are firewalled to protect an exposed demilitarized zone with hardened bastion hosts providing authorized services, monitored by intrusion detection systems (IDS), and isolated from the internal network by additional firewalls. No unnecessary ports are left open, and external network scans will find no vulnerabilities. This effectively isolates the internal systems from the uncontrolled external environment at the network infrastructure level, but at the application level things are different. By design, the Web server provides external connectivity to internal functions because that is the powerful advantage of E-commerce. However, this means that the external users are interacting with database and application servers that are not directly exposed through the infrastructure, but which may now be left exposed to attacks through the application design. The traditional approach of patching components when security vulnerabilities are found is no longer acceptable when those vulnerabilities may be discovered by attacks that disrupt databases critical to production scheduling or supply-chain ordering.

The reason for this situation is that today’s integrated business information systems are highly evolved and complex systems of interdependent components structured in a logical organization, not a piecemeal collection of independent components to be patched and secured independently. As the complexity of our systems increases, the difficulty of finding and patching all the chinks in their armor becomes unmanageable. Worse, hidden dependencies arise that prevent recognition of vulnerabilities or prevent the application of patches, as well as obscuring responsibility for maintaining security. These factors all raise the cost of maintaining application systems security, which could be mitigated by more effective consideration of security when developing application systems.

For example, many systems affected by the Structured Query Language (SQL) Slammer worm were reportedly running applications that embedded the affected Microsoft server code. Some of the system owners may not have even known their system was running the Microsoft code as a dependency within another package, which raises the question of whether they or the third-party software vendor (TPSV) bore responsibility for applying the requisite security patches. Many customers turn to TPSVs because of limited technical resources, so they are dependent on the TPSV for support, including security issues associated with TPSV packages. TPSVs cannot blindly pick up patches from platform vendors and apply them to production systems at customer sites, because of risk that the patch may cause unforeseen and undesired side effects. The cost of qualifying vendor patches and applying them at customer sites is economically unpalatable for TPSVs, so it is unlikely that they will assume this role without some prodding. Potential liability exposure might be the necessary incentive, but reducing the required expense also would reduce the disincentive. Better engineering of security as a part of application systems development could provide this reduction.

The key to engineering security as a part of the application systems development process is to see security as an inherent attribute or characteristic of systems, not a separate feature. Basically, security is a way of expressing the robustness or fragility of systems. Information security concerns are described as confidentiality, availability, and integrity. When any of those is violated and expectations or requirements are not met, it is irrelevant whether they are broken by a malicious actor or the perversity of nature. Downtime, data corruption, and inappropriate disclosure are undesirable because they cause bad effects, not because they are caused by hostile adversaries. This definition makes security a feature that should be addressed within the established application systems development community, not parcelled out for assignment to a separate organizational function. Information security practitioners can best promote improved practices by forming cooperative partnerships with application systems development organizations.

As a starting point, consider application security as a systems problem in which the overall security
requirements and results are determined by the system environment. This is really another way of saying that appropriate security is accomplished by defense-in-depth, with the defense designed into overall system structure. The appropriate security is determined by application system requirements and implemented by making design tradeoffs and utilizing underlying host and network facilities. For example, consider a sensitive application that sends user identification documents and passwords unencrypted over a highly secure network using private protocols. Conventional information security practices might argue that an environment using unencrypted passwords should not be described as highly secure, but, in light of other design features, the cost of encryption is not justified by the value. Overall, the system is sufficiently secure, although one component may be less secure than it might possibly be. The successful security practitioner must understand how much security is enough, and how to accomplish that level of security cost-effectively. Exploiting existing processes in the application systems development organization is a good way to accomplish this, and this entry offers ways to do so.

**ECONOMIC FACTORS**

In the real world of business organizations, applications are the reason systems get built and deployed, to create and promote real economic value. Management decisions are driven most clearly by economic factors in the business world, but cost–benefit analyses are the underlying decisive factors in most sectors. There are complex psychological factors involved in accepting a certain cost in order to prevent risking an uncertain cost, so justifying the costs of information security programs on the basis of risk and cost avoidance can be difficult. It seems better to understand the forces that drive business initiatives and align security program justifications in harmony with them.

The fundamental issues that motivate the need for continued improvement in applications systems in business are nontechnical in nature. Economics is always the overriding priority, because even long-term strategic initiatives are undertaken in expectation of profitable returns on the investment. This gives systems associated with direct revenue producing activities a high stature, with those involved with handling money equally important (in many but not all companies, sales is more important than finance or operations). Systems dealing with cost containment and organizational overhead are not as high a priority, which may be significant to security program investments. Competitive advantage is a significant priority, because it generates economic benefits. Managers are always under pressure to reduce costs, and schedule is a cost, so managers are also pressed to shorten delivery dates as much as possible. All of these factors work against an isolated information security program that presents a clearly measurable cost against benefits of uncertain economic value, and make it desirable to find ways to use security programs to add measurable value.

Costs of developing information systems are particularly difficult issues for most organizations, because of a number of inherent factors. Systems development is a highly specialized technical discipline that requires creative problem solving. The combination of discipline and creativity is not easily managed, leading to frequent schedule problems and associated budget overruns. Until a system is completed, the development results are not apparent, which forces management to expect success in large part based only on faith in the developers. These factors make development managers especially sensitive to issues that might affect schedule and costs. Security requirements introduce additional complexity and requirements into an already difficult development environment, so information security programs are often not embraced enthusiastically by systems developers. Using security initiatives to help facilitate meeting development schedules and budget requirements is a desirable alternative that improves teamwork.

Experience has consistently shown that the cost of fixing problems scales dramatically upward later in the application systems life cycle. Obviously, the cost of fixing a problem in design is much less than the cost of finding and fixing it once the system is built and in QA testing, and the cost of finding and fixing it once the system is in production use is even more. As a rough rule of thumb, the cost of fixing problems increases by an order of magnitude, or is about 1 times as much, for each stage later in the life cycle that the problem is found and fixed. Doing it right the first time is easiest and cheapest! This is really the fundamental drawback in the common approach to fixing security flaws as they are found in the field.

This phenomenon provides a great opportunity to turn the situation around and use security engineering to contribute positive value during the development process. By providing tools and techniques to identify and fix problems earlier in the system life cycle, security engineering can help to reduce the costs of those problems. For a simple example, buffer overruns frequently are the cause of vulnerabilities exploited by malicious adversaries, but they are also a cause of failures due to inadvertent errors, so they are undesirable because they cause a variety of problems. Thus, QA should and often does test for such scenarios. If QA is testing for buffer overruns, it will be much less expensive for developers to diligently avoid creating any that reach QA. That means using design and implementation techniques that prevent them and development tools that automatically
recognize and test for them. This simple example shows good development engineering practice as well as purely information security considerations, but it illustrates the potential value that security engineering can provide by helping to reduce the cost of developing robust systems.

One major contributor to the cost escalation as problems are found and fixed later in the life cycle is the investment in schedule resources. Personnel and equipment have associated costs that must be accrued over time, so any extension of the schedule causes an increase in costs. This is a very important point for security practitioners to consider in their interaction with development organizations, because schedule is a very important and sensitive issue for developers. Any perception by the development team that security measures might cause delays or impede schedule progress is likely to lead to an adversarial relationship between the developers and the security practitioners. On the other hand, sensitivity to schedule issues and helpful cooperation in seeking to improve schedule performance will engender a much more positive relationship. Because many of the security concerns, especially those associated with availability and integrity, are also aspects of robust, reliable application systems, promoting good information security practices will contribute to improving quality without impacting schedule.

One particular issue around schedule may be of particular concern and an especially sensitive issue for the security practitioner to consider in certain development organizations. Software developers make a distinction between software prototypes, which are “quick and dirty” implementations used to explore design alternatives and evaluate their characteristics, and production-quality code that refines the chosen design alternative into a solid, robust implementation. A frequent issue is the pressure to take software prototypes to release prematurely, before refinements such as error checking or buffer bounds checking are added. A software development methodology referred to by terms, such as “rapid deployment” or “extreme development” has gained some vogue, based on alleged cost reductions realized from dramatic schedule reductions. This methodology purports to reduce time and cost spent in development by using a quick turnaround to reduce the cost of fixing only those problems that are found to occur in production operations (the argument is “why waste time designing problems that may never occur?”). This may simply hide costs by shifting them from development to operations or applications users, which is where the effects of production problems will be borne. The security risk is that such extreme development methodologies may be encouraging bad behavior (in slighting design and QA) for schedule rewards at the expense of introducing vulnerabilities that will only be recognized when they are exposed by operational incidents. These methodologies may have value to the organization, but need to be scrutinized carefully for total life-cycle cost justifications. Security practitioners should be aware that such “bleeding edge” approaches are often extremely attractive to the creative technical personnel on development teams and that related issues (such as security compromises) may turn into political hot potatoes.

To summarize, the main factors that are the drivers for business applications of information systems are nontechnical and primarily economic in nature. Direct financial impacts, such as revenues and cost, are extremely important, and strategic issues, such as agility and competitive position, are also very significant. These needs motivate the need for applications systems and also shape the organizational environment and life cycle of such systems. Businesses will always want better systems sooner and cheaper, so anything contrary to those imperatives will be swimming against the tide. Information security practitioners need to align their efforts to promote these business priorities and position themselves in the mainstream of organizational efforts supporting those priorities in order to effectively accomplish the mission of protecting the information assets of the organization. One way to accomplish this is to take the role of collaborator and promoter or evangelist preaching value of security and cost of insecurity within the application systems development community.

**APPLICATION SYSTEMS TECHNOLOGY BASE**

It is important to remember that applications are the reason systems get built and deployed, to create and promote real business value. All the technology involved is simply a means to the end of delivering application functions to the users that benefit from their value. The systems environment, including the operating system kernel, utilities and administrative tools, user interfaces, software environments, network infrastructure and so on, is just the overhead required to deliver applications and realize the value that justifies their existence. Information systems security seeks to protect the components comprising the application systems environment for two basic reasons: 1) to keep them from being used to mount attacks; and 2) because they are needed by applications. Protecting those components is a means to the end of safeguarding business information assets, not an end in itself.

Business information assets exist within the context of information systems. Safeguarding those assets is accomplished by protecting the information systems that contain them. In seeking to do so, it is helpful to understand the nature of the information systems as well as the information assets we seek to protect. This section presents a discussion of information systems theory and
practice, focused on some features of great practical importance to applications and security.

In the most general meaning, systems are a collection of functional elements organized in structure so that they interact to perform a particular function or task. Elements are often modular subsystems that can be viewed as independent systems themselves. Thus, a distributed application system may be comprised of network elements, such as hosts and servers, that are also individual systems operating in a network environment. The view of systems as a collection of subsystems that may be considered as independent systems themselves has some very important consequences that must be understood by the security practitioner concerned with systems security.

For one, a complex networked system may be a fragile assembly of robust components, because the structure and interactions of components are essential for the proper function of the system. The common approach of fixing security vulnerabilities as they are discovered has the effect of hardening the local components at the level of the patch, but not necessarily improving the security of the systems that incorporate those components. For example, a buffer overflow attack is a way of circumventing access controls on a hardened network. Using permitted traffic to carry malicious content through the controls on secured channels, in order to ultimately exploit an implementation flaw, allows the perpetrator to break containment and obtain unsecured access on a bastion host within a secured perimeter. Arguably, the implementation flaw could be said to make the network vulnerable instead of secure, but the vulnerability could be masked by filtering malformed traffic within the network instead of exposing the flawed implementation to potentially hostile input. The point is that the network system as a whole may be more or less vulnerable, independent of any one component.

Another consequence of viewing systems as a collection of subsystems is that it creates a hierarchical relationship in which it is essential to define the appropriate level of discussion in order to establish the scope and bounds of the system entities. This is extremely important for the development process, because the most common approach to developing information systems is to define modular functions that are subsequently refined and arranged in structures of increasing complexity. Managing this process and the resulting complexity is one of the major challenges in the field of business information systems, and especially in systems development. Failure to adequately meet this challenge may be the underlying cause of most security vulnerabilities.

One approach to managing this complexity is to view the hierarchical structure of information systems in an orderly sequence from a particular perspective. Two perspectives commonly encountered are top down and bottom up. Top-down design generates abstract systems design, broken down into software subsystems of programs and data structures. Bottom-up construction assembles physical resources into networks that run programs and communicate data. The software engineering process designs application systems from the top down and builds them from the bottom up.

Another way to express this is to consider that automated information systems exist at the intersection of a top-down perspective that describes abstract logical design and a bottom-up view of concrete physical implementation. The top-down approach deals with functional business information systems (e.g., payroll, order entry, etc.) and the bottom-up approach deals with programs and data on networked hardware and software systems.

This creates an ambiguity that commonly leads to confusion over which view is meant when referring to systems, for example, identifying systems for a security assessment. Do we mean the logical business function or the software and hardware that implement it? Evaluating access controls on a distributed ERP application is not the same as evaluating access controls on the networked servers hosting it. The security practitioner must clearly understand and communicate which perspective is intended when the context does not sufficiently identify the reference to make it unambiguous.

Information security practitioners need to take both views into account. Effective security programs must consider the value at risk, which can really only be determined based on the business functions expressed in the top-down perspective, and the cost of protecting the information assets, which depends on the implementation details embodied in the bottom-up view. The challenge is to secure applications by incorporating security as an integral part of the engineering process that develops and integrates both the top-down design and the bottom-up implementation of application systems.

There are also two phases of an application system’s life during which different security concerns should be considered. Most commonly, application systems security is focused on the application during production operations, as this is when the application is performing its function of generating value (and thus, where it spends most of its lifetime). The development of application systems is generally considered separately, more as a production application of development tools and systems than in the context of the application being developed. This may minimize several important concerns. For one thing, security breaches during development may disclose or introduce vulnerabilities in the application itself (“dumpster diving” is an exploit that may target development documentation to identify vulnerabilities to be attacked in the application system product). For another thing, the development process
may interact with production operations during design, testing, and deployment in ways that create or expose vulnerabilities in the production environment. For those reasons, application development should be considered in conjunction with the operational application systems by security practitioners concerned with the security of such systems. This is particularly challenging because the nature of development organizations and activities is distinctly different from production operations. It may be best to avoid tackling security issues in the development environment head-on and instead cooperatively team with developers to focus on improving security of the resulting application systems, while also seeking to indirectly improve development environment security (awareness and influence will be more effective with the developer personalities than with direct authority).

APPLICATION SYSTEMS COMPONENTS

Application systems may be comprised of a tremendous variety of components or subsystems, each of which introduces its own particular issues and concerns regarding security. In addition, the relationships and interactions among components also introduce further security complications. Developers who might be ignorant of security considerations may overlook or underestimate the importance of these issues. The security practitioner should be aware of the nature of major components that frequently comprise application systems, and have some acquaintance with the security issues that might be associated with them.

A superficial survey of the various components associated with applications systems is provided in this section, as an introduction to the many aspects that need to be considered both by application developers and security practitioners. The full range of components potentially comprising application systems includes hardware and firmware, operating system components (kernel, drivers, memory management), process management software (loader, scheduler, termination handler, core dumper), file system, command interpreter (shell), utilities, system runtime environment (environment variables, ports, configuration parameters), network protocol stacks, database software (e.g., SQL), user interfaces (graphical user interfaces [GUIs], command shells), help systems, runtime systems (language support libraries, object management systems), development tools (compilers, source management tools, profilers, debuggers, linkers, diagnostics), console management tools (backup utilities, remote administration packages, configuration management and remote deployment facilities, load managers, event loggers, tools, user account managers), and the organizational environment (management, operations personnel, users, developers, vendor support staff, etc.).

The foundation for any system is the hardware used to implement it. Unfortunately, there are often features designed into the hardware to support security that are not utilized within the systems and application software. Sometimes the features are ignored by the software environment; others are more or less fully supported by the basic system software, but hidden or unutilized in other software components. Some hardware provides extremely flexible features that are normally utilized in a standard fashion, but can be used in other ways. This may camouflage security risks, because many users and technical staff may be unaware of the potential for alternate usages. An example is network interface cards (NICs) for Ethernet, which implement a media access control (MAC) address that is hard-coded by the manufacturer and encodes the manufacturer ID. However, the Ethernet chips used in some NIC cards allow the MAC address to be set to other arbitrary values by running software, which could introduce unrecognized security vulnerabilities in some systems.

Most intelligent hardware devices employ embedded firmware implementing the necessary system processing and control features. In the case of stand-alone network hardware, this firmware may embody the entire special purpose operating system required to install, configure, operate, maintain, and manage the device. General purpose computers incorporate firmware to extend basic hardware functions; for example, the NIC card MAC address functionality previously described is implemented by a combination of hardware and firmware. Differing firmware revision levels may introduce inconsistent security features, either fixing previously discovered vulnerabilities or introducing new ones. (A pseudo-scientific law of computer programming states that fixing any bug simply replaces it with two smaller bugs!) Firmware configuration management introduces potential security vulnerabilities. An example of the security vulnerabilities associated with firmware features would be the viruses that rewrite the firmware in the boot ROM to substitute virus code.

Operating system software provides functions to extend the basic hardware environment to provide more conveniently usable features for general purpose uses. The major operating system software consists of kernel implementing I/O facilities, memory management, CPU scheduling, device drivers, file system code, and process management (loader, scheduler, termination handler, and perhaps a core dumper). The basic facilities to support user authentication, authorization, and access control, or privileges and protections, are provided by operating systems functions. In addition, the associated command interpreters (or shell) and utilities may be considered part of the operating system, although the distinction between bundled and unbundled system components
becomes very indistinct in this area. This feature is often exploited by intruders who replace bundled system components with modified versions to cover their tracks or introduce additional vulnerabilities. The operating system environment is often considered as separate and distinct from applications systems components, although it really is an essential element determining the fundamental security characteristics presented to the application system. Many security problems result from attacks that exploit vulnerabilities in applications or utilities to break out of the software function, to gain access to unintended and unrestricted operating systems capabilities. The capabilities exposed to such exploits are determined by how the application systems developers have utilized the underlying operating system features, but generally they are very significant concerns for the security practitioner.

Network protocols are an essential element of distributed systems, generally following the layered architecture made famous by the ISO Open System Interconnection protocol stack model. Internet protocols based on TCP/IP have become ubiquitous, but other protocols models still are used, although less widely. Many older protocols that once used an entirely proprietary protocol stack have substituted TCP/IP for lower layers while retaining their distinct higher-level functional interfaces. There are many security concerns associated with network protocols. The criticality of their functions and their nature as communications media make them especially attractive targets for attacks, both as an end objective (e.g., denial of service, data theft) and as a stepping stone (e.g., worm vectors, relay systems). Because of this, network security is a separate specialized field, but the dependency on network protocols by distributed applications systems forces consideration of protocols as an important factor relevant to application security. The tight integration of network protocols with local I/O in some modern operating systems makes it easy to inject malicious input from remote sources. This is exploited by attacks, such as relatively low-level buffer overflows and higher-level cross-site scripting attacks. Network protocols are extremely flexible and must be carefully considered for potentially dangerous interactions with applications systems. This is one reason that it is imperative to ensure that any protocols received by a system must be properly handled (i.e., no unnecessary open ports listening for TCP/IP input, and all services on required ports properly configured for security).

GUIs are commonly used for interactive applications, utilities, and commands in modern systems. It is important to keep in mind that many systems incorporate software that uses command line interfaces, either because they were developed before GUIs were so common (legacy code), or because command lines are more convenient for expert users and automated scripting. Such hidden nonGUI interfaces may provide targets for attackers, especially using network protocols to inject malicious input. Developers of new programs providing such interfaces for scripting convenience may assume that all input will come from local (and thus trusted) sources, and therefore not provide careful input validation and buffer checking, thus creating potential vulnerabilities to remote attackers or malicious local users. Because system designers frequently differentiate user interfaces as front-end GUIs from back-end processing of application business logic, this should be an area of particular concern for application systems security.

Database software, such as SQL processors, is an essential component of many application systems, and, as such, must be a major security concern. SQL packages may themselves be subsystems, including multiple components, and the interaction between these components may have important security implications. For example, the SQL Slammer worm exploited vulnerability in an SQL component interface in order to cause malicious commands to be executed by other system components. This vulnerability was present not only in stand-alone SQL servers, but also in embedded database components hidden within packaged application systems.

There is a help system provided with most modern application systems and GUIs, to provide context-specific assistance to the application users. This is not normally considered a security concern and has not been an attractive target for exploits. There is a slight possibility that the components used to provide application help could have vulnerabilities that might be subject to some attacks, but this seems fairly insignificant. A more significant concern might be the potential for inappropriate disclosure of information through context-specific help facilities, especially if the help facilities also provide an interface to remote diagnostic and support tools. In general, this area is probably not a major application systems security concern, but at the same time it should not be completely forgotten.

The run-time execution environment within a system consists of the various parameters that are used to set variable values controlling system functions; for example, the IP address of a networked host. Many of these configuration parameters are stored in some nonvolatile format (e.g., parameter files) and then used to initialize values for dynamic elements of the system. The configuration files may be read and interpreted by a script processor (e.g., through the command shell) or directly by the associated program itself. Sometimes the values are stored in environment variables to make them accessible over a longer period of time within the executing system environment. The contents of environment variables and configuration files are subject to attack and may provide avenues for exploits. These features are
provided by the operating system and are subject to whatever access controls are implemented in that system and used by the developers of the particular features. An important issue regarding system privilege and protection mechanisms is that developers often find finely granular mechanisms cumbersome and inconvenient and thus may use shortcuts, such as elevated privilege or less protection, to reduce implementation efforts at the expense of security. Such features are usually considered internal details that are not exposed to external threats and thus may not be protected beyond “security through obscurity,” which may leave vulnerabilities such as the potential for scripts to inject malicious commands (frequently executed with elevated privilege or undesirable account context). Also, inappropriate modification of these component values could well result in denial of service. The application systems security concerns associated with these features are certainly significant, but the relative obscurity of any vulnerabilities helps to moderate the priority of those concerns.

Modern software engineering seeks to abstract logical representations of function from the concrete (albeit virtual) resources used to implement those functions. As a consequence, application development tools such as object-oriented environments include extensive runtime support, which is often hidden even from the application developers. From a software engineering perspective, this is desirable as a means of hiding complexity, but from a security perspective this has the undesirable consequence of hiding dependencies and possible vulnerabilities. Object reuse is a major priority for reducing development costs, and this requires the most general and least constrained implementations. As a result, bounds and value checking may be compromised or complicated because the specific validation requirements often depend on the particular usage. It is not possible to effectively perform some validation (such as buffer size) external to the module or object using the values, but it may be more complicated to implement an effective check at the site of usage for arguments supplied externally by an invoking object or module. The security concerns in this area seem to be primarily focused on denial-of-service possibilities, although there should also be some awareness of dependencies on external vendors to provide secure components and eliminate vulnerabilities in their object management and compiler runtime systems. A related area of concern is the use of dynamic linked libraries in some systems, which provides a potential vulnerability for substitution of components incorporating malicious code in place of the original trusted components. This could be utilized by “root kits” installed to further exploit a compromised system. Application systems would be vulnerable to this exploit, although it may be more likely to target bundled host system components that are more widely known to attackers.

Management and operational support tools are essential components associated with any significant application systems, especially in a distributed network environment that may use “lights out” data center practices. The phrase “lights out” refers to data centers running 24/7 without being staffed 24/7, depending on automated management tools to allow remote administration by remote operations centers with online monitoring, or on-call operations personnel alerted using pagers. Event loggers, reporting and filtering tools, centralized monitors, and remote access to management consoles are all elements of the management systems used to support online operations for network systems delivering critical applications. These components are especially critical because they are vital to maintaining security of applications systems and are complex and subject to vulnerabilities themselves. The good news is that management systems are frequently supplied by major vendors who recognize the critical role of such systems and are committed to their security. The bad news is that such powerful management systems may introduce vulnerabilities, especially to application dependencies (the most common denial-of-service attacks are those inadvertently perpetrated by system and network administrators making mistakes during routine operations). Other management and operational support tools include backup utilities, load managers, deployment and configuration management tools, and user account managers. Such tools are obviously significant security concerns, but those concerns may not have received the same scrutiny as isolated functional utilities as they do for centralized console managers. For example, in small organizations or for less-visible applications, backups may be routinely performed but never tested. Failure modes need to be considered as potential security issues, so that a network glitch during a remote upgrade does not result in a complete denial of service (such considerations highlight the indistinct boundary between security and application design and implementation). The security practitioner concerned with application systems security needs to be very aware of and concerned about these tools, and may want to enlist operations and development staff to cooperatively review and address security implications in these areas.

As previously mentioned, applications systems development presents a unique environment with its own set of security considerations. Development tools include source management packages, compilers, linkers, profilers, debuggers, diagnostics, and many other utilities. In addition, developers and QA testers may need the ability to manipulate the running system environment in ways that production operations and ordinary users do not require (e.g., to set up or recover from specific test scenarios), and thus may be routinely granted access to use privileges that present security concerns. Because of this, development systems and accounts may be
particularly attractive and valuable targets for attackers. There may also be vulnerabilities exposed in the development environment and process that are not present in production operations; for example, if samples of production data are used for testing without ensuring that appropriate protection is provided for sensitive content. This problem may be exacerbated once applications systems move to production, because problems during production may require access to sensitive data or even to production systems. Normally, a well-managed development organization will be effectively isolated from production to minimize security exposures, but this discipline comes at a cost and is especially subject to compromise when problems occur. Such situations require heightened awareness of security issues by all personnel involved (and, of course, entail a heightened stress level that makes everyone less receptive to reminders, highlighting the importance of cultivating routine awareness of good practice).

Finally, no application system functions in a vacuum. Application systems exist to serve human purposes in some form or fashion. The interactions with humans occur within an organizational environment and culture that defines the fundamental security context that must be considered by any effective practitioner. The organization includes management, users, operations personnel, developers, and external personnel, such as vendor support staff. Each has their own function and may place their job as a higher priority than security, so it is human nature that they may take shortcuts for convenience or intentionally or unintentionally compromise security in other ways. The security practitioner must remember that the goal of security is to protect the utility of systems to the organization, which requires promoting awareness of security considerations by all personnel. Most importantly, the practitioner must remember that the greatest utility is likely not the most secure system, but one with carefully considered security policies and practices that are appropriate to the system and organization. The reason for cooperatively integrating application systems security concerns into the development process is to properly establish the most appropriate security posture and to effectively implement it.

**TECHNICAL CONCERNS FOR APPLICATION SYSTEMS**

Some specific technical areas frequently cause security issues within application systems. This may be caused by the characteristics of the technical features involved (difficulty of use or complexity of feature), the nature of the use, or the limitations of application developers. Some particular concerns are input validation (filter for illegal values as well as protecting for buffer overflows), memory management (especially buffer overflow protection, but also stale data violating confidentiality, etc.), authentication/authorization/access (AAA) control (application implementations often trade strength for user convenience), session management (HTTP is stateless, so cookies are used to provide persistent context with extremely weak AAA), and configuration management (change control and QA to prevent insecure software in production). Security practitioners need to focus attention on these issues during design, development, and testing, to avoid the costly problems surfacing later in the life cycle. Designing sound solutions in these areas will help make implementation and testing easier, benefiting the entire team.

Application packages provided to third parties (including separate organizational entities within the same corporate umbrella) should specifically identify dependencies on platform and external package features in sufficient detail to understand security issues associated with those dependencies (including but not limited to potential denial-of-service attacks). Application providers should disclose such details and their clients or customers should insist on disclosure. Internally within development organizations, engineers should document, test, and monitor security of all dependency interfaces.

**APPLICATION SYSTEMS DEVELOPMENT LIFE CYCLE**

The existence of such application systems follows a very well-understood life cycle, initially determining and specifying functional requirements for the system to be implemented. This initial functional design phase moves into an implementation design phase, which determines the technical details that will be used to implement the system. The implementation design proceeds into a development process that further refines and arranges details of technical components to create the requisite functionality required by the initial functional specifications to answer business requirements. There is an iterative process of development and testing for both individual components and the entire system as implementation progresses, to assure satisfactory quality before release for production operations.

When the QA function determines that testing has found that requirements have been successfully met for satisfactory production operations, the application system is released for deployment to production. This stage of the systems development life cycle is sometimes called release engineering, for obvious reasons. Production deployment may be a simple transition of starting to use a new system, or it may require a very extensive process of parallel testing and progressive migration of
critical functions onto the new implementation with provisions for falling back to previously used systems in the event of problems. The deployment into production requires updating configuration management systems used to control production systems, and often uses automated tools to install the appropriate configuration on production systems automatically. There may be provisions for backing out of releases, especially in extremely critical production operations, to ensure that any new release does not cause unforeseen problems (e.g., the scale of production traffic may be difficult to reproduce in QA, leaving the potential for unrecognized problems caused by volume over time).

Upon the ultimate completion of production deployment, the application system enters routine production operations and maintenance. During this phase, requirements may evolve (e.g., rules for regulatory compliance may change slightly) and new or unusual situations may reveal flaws in the design or implementation that were not caught before release. These occurrences will require some maintenance upgrades to the production application system, so production operations are often referred to as the maintenance phase of the system development life cycle. Any changes will normally require appropriate testing before release, and should follow release engineering procedures similar to major new systems.

Security practitioners concerned with disaster recovery and business continuity planning need to be especially interested in the interaction of release engineering and deployment with configuration management and console operations tools. One powerful motivator for automating configuration changes and management is the impossibility of recovering to an unknown configuration following any disaster! On a less dramatic scale, problems affecting routine system updates can have a costly ripple effect if the recovery from problems interferes with continuity of routine business operations. For example, if a network glitch interrupts the routine deployment of an automated update to a production server, the server may be left in an insecure state or simply unavailable until manual intervention restores a serviceable configuration. Preventing such situations (or recognizing and remedying them) is an opportunity to add value that will be beneficial to the entire organization.

Ultimately, the cycle ends when changing business requirements or technology motivate replacement or major enhancement of the production application system, and a new development cycle will be initiated, with deployment of the new system leading to replacement of its predecessor. Sometimes the functions provided by the application system will no longer be needed and the retirement of the system will not include any replacement. This situation can lead to legacy systems becoming unused and forgotten but not removed, with an increased risk that inattention will lead to insecurity.

INTEGRATING SECURITY INTO THE SYSTEMS LIFE CYCLE

The introduction to this entry discussed the historical approach of information security programs, focusing efforts and resources bottom up, on technical components rather than taking a holistic systems-oriented view of the problem. This approach is appropriate during the operational phase of the systems life cycle, but as the discussion about economic factors showed, retrofitting security with patches after system deployment is woefully expensive as well as fundamentally ineffective because of the nature of systems themselves. The paradigm shift suggested at the beginning of this entry focuses on integrating security into all phases of the systems development life cycle as a way to provide more cost-effective improvements in application system security.

Treating security as a separate issue assigned to an isolated organizational unit creates a situation in which the security function too often ends up the antagonist of developers in the application systems development process. Because the development team goal is to ship the product as soon as possible, imposing security requirements on the implementation design seems a costly impediment to achieving that goal. However, as we have seen, the development team and the information security practitioner share a common interest in deploying robust systems, because availability and integrity are fundamental requirements for a functional system. Confidentiality is also a common interest, but based on separate business issues of competition, compliance, customer care (or privacy), which might be called the “four Cs” of confidentiality.

Benefits from including security in the entire system development life cycle start with the early top-down engineering design process, by helping to design robust systems more cost-effectively. As previously discussed, system development economics benefit greatly by meeting requirements earlier in the development process instead of reworking designs to fix shortcomings later. Presenting security requirements as metrics of robust quality early in the process motivates good practice in a cooperative rather than an antagonistic fashion. Throughout the development process, security considerations can be used to focus attention on critical aspects of the application system to improve product quality while avoiding costs for later patchwork. Overall, security can be an enabler of better performance by development teams, improving quality without
impacting schedule, by better identifying and addressing critical concerns affecting robust quality.

Different stages in the application systems development life cycle have different security requirements and present different security challenges. Requirement documents and functional specifications are frequently housed on centralized document management or groupware systems, so security administration is not particularly challenging. Development hosts often present a particularly challenging technical environment, because creative systems developers are often inclined to push the limits both organizationally as well as technically. There is often friction between system administrators responsible for development systems and the developers using those systems, especially when powerful desktop workstations are used to facilitate development in a centrally managed network environment. Systems used for testing and quality assurance are usually much more cut-and-dried in their security requirements, because they normally should use environments identical to production as much as possible (exceptions should be clearly justified, perhaps by test management toolset requirements).

Deployment, or release engineering, is the interface and transition between development and production. Because they are responsible for moving system packages that have completed testing into production, security is a routine concern to which the users of these systems are well attuned. The security practitioner should keep in mind that these systems may not be monitored in the same way that production operations are monitored, although they would be high-value targets for an adversary seeking to inject malicious code into the production environment, or to simply disrupt production by causing unserviceable components to be released. Also, careful management of deployed configurations is an essential requirement for successful disaster recovery efforts, because it is impossible to recover to an unknown configuration.

The operations phase of the systems development life cycle is the usual focus of information security programs, so it is regarded as outside the scope of this entry except for one aspect. Failures occurring during production operations may require unusual diagnostic or emergency maintenance activities that force exceptions to normal operational security practices, or involve development or vendor personnel. These situations may cause unforeseen security implications, such as the potential exposure of confidential information contained in diagnostic files (e.g., core dumps) transmitted outside the normal security perimeter. Pressure to get corrections into production may lead to compromises in security, and such issues need to be carefully managed to ensure that such compromises are appropriate and not just convenient.

Security practitioners may find that system administrators and development managers share concerns over systems security issues, especially for development systems, and the most effective way to address those security concerns might be in the guise of organizational issues within the development team. For example, developers that use elevated privileges to bypass access control mechanisms during implementation may inadvertently introduce dependencies that are inappropriate to the production environment. These are subtle and costly problems, because they may not be discovered until much later in the QA process, or even after production release, necessitating costly correction efforts. Aligning security concerns with project management issues in this way, allows the practitioner to develop a recognition of the security function as supporting important values for the entire application systems development organization.

One way to classify security vulnerabilities is to identify the stage in the systems development life cycle in which the vulnerability is created, as a way to help in focusing appropriate attention on correcting vulnerabilities. This also allows defect tracking to assign responsibility if a flaw is discovered in the implementation. For example, input validation should be considered a design requirement, and thus included as a part of the functional specifications implemented in development. QA testing is commonly driven from functional specifications, so the discovery of a vulnerability because input validation is lacking might be a specification failure or a combination of implementation and testing failures. This feedback can be used for process improvement within the development organization, and may often be provided by defect tracking tools. Integrating security concerns into this feedback process is a way to align security efforts with the organizational efforts to continuously improve the development process and results.

**INFORMATION CRITICALITY MATRIX TOOL FOR SECURITY EVALUATION**

Disclaimer: The National Security Agency has neither reviewed nor approved the following material. It is purely the author’s understanding of material obtained from a variety of sources, and his logical extensions of that material.

The InfoSec Assessment Methodology (IAM) developed by the National Security Agency (NSA) provides many useful features. One element of the IAM is particularly promising as a tool for improving application systems security and providing benefits of value to development schedules and results. This section will summarize the IAM, introduce the Information
Criticality Matrix used in the IAM, and suggest extensions of that matrix for use in application systems development.

One of the roles for the NSA is responsibility for information assurance for information infrastructures critical to U.S. national security interests, through the Information Assurance Directorate (IAD). One NSA/IAD program is the InfoSec Assessment Training and Rating Program (IATRP). According to the NSA Web site (http://www.nsa.gov/isso/iam/index.htm), NSA developed the IATRP, a two-part (training and rating) program, for the benefit of government organizations trying to raise their InfoSec posture in general or specifically trying to comply with the Presidential Decision Directive-63 requirement for vulnerability assessments. The IAM is a detailed and systematic way of examining information security programs.

The IAM framework specifically provides for customized extensions to accommodate particular situations having needs that do not fit or go beyond the standard IAM requirements, with the provision that any modifications do not reduce the level of assurance required to be IAM compliant. Much of the IAM codifies accepted practices, describing project organization, standard activities, required elements, and minimum performance expectations for acceptable results. A key feature is the use of a matrix to identify information and systems and structure measurement of the criticality of security for those components. Consistent with common information security practice, the IAM is primarily focused on the needs of operational organizations and their processes rather than their downstream products. This entry proposes extending the framework and techniques used in the IAM by applying them in coordination with the application systems life cycle.

To summarize the IAM, it provides a framework for projects evaluating information systems security programs. The purpose is to review the information system security posture of a specified operational system to assure that the security program is appropriate for the system requirements. It does not encompass technical vulnerability assessments, such as penetration testing or network mapping. There are three phases to the IAM: 1) the preassessment phase; 2) an onsite activities phase; and 3) a postassessment phase. The preassessment phase entails project planning and preparation, including organizational agreements, establishing the scope and bounds of the project, reviewing information about the systems being assessed, reviewing existing security program documentation, and planning and preparing for the onsite activities. The onsite activities gather data to explore and validate information from the preassessment phase and provide initial analysis and feedback to the organization responsible for the systems being assessed. The postassessment phase finalizes the analysis by incorporating results of the onsite activities with information provided during the preassessment phase, and produces a final report.

The IAM specifies a set of baseline categories that are normally reviewed by a compliant evaluation project, unless particular items are specifically excluded by agreement with the assessment client. Any categories that are omitted must be identified and justified, with the requirement that the omission does not reduce the level of assurance provided by the assessment. The standard IAM baseline information categories are InfoSec documentation, InfoSec roles and responsibilities, identification and authorization, account management, session controls, external connectivity, telecommunications, auditing, virus protection, contingency planning, maintenance, configuration management, backups, labeling, media sanitization/disposal, physical environment, personnel security, training, and awareness. Additional categories may optionally be added to accommodate specific requirements of the particular systems being evaluated (e.g., encryption), or to provide finer granularity. For example, incident response might be considered part of InfoSec roles and responsibilities and intrusion detection might be included under auditing, or they might be broken out as separate categories.

The purpose of the IAM is to ensure compliance with federal law mandating appropriate security for automated information systems at sensitive but unclassified (SBU) level or above. One purpose of the preassessment phase is to “identify subject systems, including system boundaries.” This requires addressing both logical and physical systems, along the lines discussed in the section of this entry discussing application systems technology. Because a logical application system may encompass many physical systems, each of which processes a subset of the system information, it is very useful to have a means of establishing the security requirements for each individual component of the system. The subset may be a particular piece of information or a particular piece of physical equipment. In practice, the security requirements are determined by the nature of the information involved, so the equipment security requirements are derived from the security requirements of the information processed by the particular equipment. The “information criticality matrix” is a tool invented by Mr. Wilbur J. Hildebrand, Jr., NSA’s Chief of InfoSec Assessment Services, for use in the IAM to determine the security requirements for particular items of information.

The “information criticality matrix” structures the determination of information security requirements by listing the information elements within the logical system and associated impact values for security attributes. The IAM uses confidentiality, integrity, and availability as the three required standard attributes, and requires that any change to this list be clearly documented. For example, one potential addition might be
nonrepudiation, and it would be appropriate to justify the requirement for including it as a separate critical attribute. The result of this matrix provides an initial determination of information security requirements for the overall system, and also values to be used in further refinement of security requirements. The first refinement is the analysis of logical subsystems by selecting the entries for the specific information handled by those subsystems and using them to determine information security requirements for the subsystem. Another refinement is to determine the information security requirement for physical components, based on the information security requirements of all the information (or subsystems) processed by the component. These refinements provide the basis for evaluating whether the information security programs for the affected systems are appropriate for the security requirements of the information contained therein.

### CRITICALITY MATRIX USE IN APPLICATION SYSTEMS DEVELOPMENT

The IAM criticality matrix provides a tool for initially determining information security requirements from a top-down logical systems perspective and then deriving security requirements for the bottom-up systems implementation. This can be productively applied to the development of application systems in several ways. One powerful extension would be to generalize the information resources evaluated using the criticality matrix to include functional processing components within the logical system design, so that the importance of particular software modules can be determined. This not only serves to focus security requirements, it provides value of great benefit to the systems development project in general, because availability and integrity measure, not just security requirements, but overall importance for the particular functions evaluated. The ability to better measure the importance of functional modules is very beneficial for the systems development project in general because it helps to guide project planning and management in areas, such as resource allocation, design attention, testing requirements, defect tracking, etc.

Another use of the criticality matrix to integrate security engineering into the application systems development process would be to focus more attention on addressing technical vulnerabilities (such as buffer overflows) in areas where they would affect critical components versus areas that are relatively less critical. In some environments, this might help guide management decisions about whether rapid prototyping is an appropriate tool or whether critical components might require additional development attention to ensure appropriate production-quality systems are released for deployment. This provides another opportunity for security practitioners to develop a cooperative relationship as productive contributors generating value important to the application systems development team.

The criticality matrix could even be used to analyze the information security requirements of an application development project over the course of the system development life cycle, and thus to better focus efforts to provide appropriate security for systems used by development projects. Security requirements for systems housing functional specifications and design documents will be different from those of systems used for implementation development, testing, or deployment; and some of those security profiles may be different, depending on the security requirements of the application systems involved. The criticality matrix provides a tool to facilitate consistent evaluation of those security requirements, so that the development projects are neither burdened nor exposed inappropriately.

The criticality matrix can be used in different ways during different stages of the systems development life cycle. During application systems design, it can be used to set security and quality requirements for project features and for project planning and management. During development, it can be used to set appropriate standards for production implementation quality, source management, and feature completion. During QA, it can be used to focus test efforts most effectively, design test strategies, determine the scope and coverage of testing, and track defects according to importance and priority. In operations, it can guide configuration management and deployment planning, and rollout; prioritize bug tracking; and map defects into the systems development life cycle quality and security matrix to provide feedback for process improvement.

### FUTURE DIRECTIONS

This entry has surveyed some information security considerations pertinent to application systems development, reviewed a number of areas related to application systems and the technical and organizational development environments, and described a novel tool for incorporating security engineering into the application development process. In the course of these topics, several suggestions for subsequent research and development were mentioned. This section reviews some possible directions for subsequent efforts.

There are a number of automated tools in use for managing systems development projects, automating testing, tracking defects, and configuration management and deployment. Incorporation of support for security engineering facilities, such as the criticality matrix,
could be a useful enhancement to such tools. Similarly, intrusion detection systems and management console tools used for systems and network administration of production operations could be enhanced to use the IAM criticality matrix as a factor in prioritizing alerts for all events based on system criticality. It seems especially useful to have configuration management systems provide alerts for discrepancies, and management consoles to report those alerts, with severity settings keyed to the criticality of the subject system, as an adjunct to other IDS monitoring facilities. Undoubtedly, experience will suggest even more and better possibilities in the years ahead.

ACKNOWLEDGMENTS

The author would like to express grateful appreciation and thanks to Wilbur J. Hildebrand, Dr. Peter S. Tippett, and Jerry Weinberg.

REFERENCE


BIBLIOGRAPHY