Introduction to System Strategies

Robert J. Ellison

Carol Woody

May 2007

ABSTRACT: Trustworthiness can no longer be predicted by building software systems from discrete, isolated pieces that address static requirements within planned cost and schedule. Each new or updated component joins an existing operational environment and must merge with that legacy to form an operational whole. Today's technology must support an operating environment that is driven by business goals and organizational needs instead of a predefined infrastructure that functions within established technology constraints. The operating environment can be geographically and managerially distributed and dynamically changing. Few businesses can stop to make changes and then restart. This introduction discusses the effects of the changing operational environment on the development of secure systems.

TRENDS AFFECTING SYSTEM SECURITY

The expanding scope, complexity, and scale of current and planned systems influence the ways in which we must address security and drive a need to reassess the development assumptions that we successfully applied in the past. A number of trends will influence how we need to address security.

- Instead of centralized control, which was the norm for large stand-alone systems, developers will have to consider multiple and often independent control points for systems of systems.
- Increased integration among systems has reduced the capability to make wide-scale changes quickly. In addition, for independently managed systems, upgrades are not necessarily synchronized. Services shared by multiple systems have been introduced to reduce redundancy and improve interface manageability. However, we need to maintain operational capabilities with appropriate security as those services are upgraded or retired and as new services are added.
- With the increased integration among independently developed and operated systems, we will have a heterogeneous collection of components, multiple implementations of common interfaces, and inconsistencies among security policies as systems and organizational policies adjust over time to changing organizational needs.

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

- System development increasingly has to consider how users and operators contribute to the overall behavior of the system. We no longer have a distinct boundary between people and systems [SEI 06].
- With the erosion of the people/system boundary and the mismatches and errors introduced by independently developed and managed systems, failure in some form will be more the norm than the exception, further complicating the creation and validation of security requirements for effective software and system design.

Some of these issues were raised by The Committee on Information Systems Trustworthiness that was convened by the Computer Science and Telecommunications Board (CSTB) of the National Research Council (NRC) to assess the nature of information systems trustworthiness and the prospects for technology that would increase it. Their report was issued as the document Trust in Cyberspace [Schneider 99]. Their report is an excellent summary of the issues and the research required to address them.

System-level trustworthiness requirements are typically first characterized informally. The transformation of these informal notions into precise requirements that can be imposed on individual system components is difficult and often beyond the current state of the art. Whereas a large software system such as an NIS [networked information system] cannot be developed defect-free, it is possible to improve the trustworthiness of such a system by anticipating and targeting vulnerabilities. But to determine, analyze, and, most importantly, prioritize these vulnerabilities, a good understanding is required for how subsystems interact with each other and with the other elements of the larger system—obtaining such an understanding is not possible today.

NISs pose new challenges for integration because of their distributed nature and the uncontrollability of most large networks. Thus, testing subsets of a system cannot adequately establish confidence in an entire NIS, especially when some of the subsystems are uncontrollable or unobservable as is likely in an NIS that has evolved to encompass legacy software. In addition, NISs are generally developed and deployed incrementally. Techniques to compose subsystems in ways that contribute directly to trustworthiness are, therefore, needed.

LIMITATIONS OF CURRENT TECHNIQUES

Both the technologies used and dynamic nature of the operational environment raise software risks that are typically not addressed in current practice. Current security verification approaches are primarily point-in-time strategies focused on selected pieces that are not easily adapted to the dynamics that software now has

to address. Product component accreditation is focused on a point in time and a specific instantiation of a product, which is only useful for an implementation that closely matches the one used in the accreditation effort. With the range of usage available and the frequency of upgrades for most products, the likelihood of a match is minimal. System certification assumes a "hard" system boundary under the control of a single management point and validates that the security controls within this boundary are functioning as planned, which ignores all of the system-of-systems interoperability and management issues. Vulnerability analysis evaluates an operationally ready network, system, or software set against previously identified and analyzed defects and failures at a given point in time for a specified configuration. Such techniques are of limited value when the system can be dynamically configured to meet changing operational and business needs. Additionally, for software under development, the operational context is typically not sufficiently detailed to apply any of these current techniques until system integration testing, very late in the development cycle.

Individual software components and systems operating within a system-of-systems environment cannot be evaluated for security effectiveness without considering the operational and organizational environments within which each must function. Security responsibility is distributed across the people, practices, policies, and technology. Few techniques look beyond the technology, and this organizational context is a key driver for security risk. Demonstrating regulatory compliance for Sarbanes-Oxley, HIPAA, or FISMA, which addresses only a portion of software security risk, requires effectiveness of organization controls as well as of software implemented controls. Effective security requires a careful balance among the following four areas:

- organizational management policies and procedures
- user management and practices, which includes authorization and authentication
- software development and acquisition, which includes built-in security capabilities and software reliability
- operational security practices and management

SOURCES OF COMPLEXITY

One objective for this article is to discuss the affects of the business demands on security and suggest some strategies that may help in managing the complexity. Some of those strategies represent work in progress by organizations now con-

fronting these problems and were synthesized from presentations and discussions at various conferences.

The oft-repeated adage is that complexity is the enemy of security, but complexity in modern systems is a given, and we have to manage it rather than unnecessarily adding to it. The complexity is an aggregate of technology, scale, scope, operational, and organizational issues.

Business Drivers

The quotation from the CSTB report included in the introduction for this article captures the technical challenges as the security context expands from components, to systems, and then to systems of systems. The technology concerns are only part of the problem. The development and operation of large software systems require balancing a spectrum of forces. The technical forces, for example, are reflected in the difficulty of implementing a particular function or with meeting the quality measures for reliability, performance, and security. The organizational forces include regulations such as Sarbanes-Oxley and HIPAA, integration of financial, administrative, and manufacturing control systems, effects of distributed operations, rapid pace of business change, increased business expectations for recovery with less tolerance for mistakes, expanded and more permeable system perimeter to meet business needs, and continued pressure to reduce IT costs. These business forces are transformed into technical factors. Regulations increase the business expectations for privacy and information protection that must be reflected in the fielded system (or system of systems). The expanded system perimeter increases the risks associated with vendors, suppliers, business partners, and customers. The integration of financial, administrative, and manufacturing control systems typically requires interoperability across multiple computing platforms with differing risk profiles and implementations of security policies. The pace of business change generates requirements for distributed management and system flexibility and ease of evolution, and operations increase the need for global identity management, strong access control, and accountability. Non-stop operations require that some significant changes can be made without shutting down the systems.

Security of Interoperating Systems

The perception is that technology is an enabler for organizational expansion but the costs are not understood. We have had rapid expansion of mobile computing as well as increased integration among business systems. Technologies such as web services should enable a more rapid deployment of distributed systems. Each segment is constructed and validated independently allowing for easier and faster deployment, but this can lead to operational complexity in terms of the difficulty of managing the systems that were built separately but must interoper-

ate at execution. We can deploy but the systems are increasingly likely to have hidden risks, especially in problem identification and correction, that do not appear until the systems are actively in use. Complex functions and interactions make it impossible to identify and validate every possible combination before implementation.

Stakeholder Diversity

Stakeholders who define the business needs for a new application or component can represent a highly complex range of organizational needs; they can be organizationally distributed and diverse, with conflicting, complex, and incomplete requirements. As more organizational functions are linked to share information, the technology that supports those functions is integrated to share data and support cross-functional activities. When the choices made by previously disconnected stakeholders are incompatible, poorly planned integration can leave gaps that provide opportunities for security problems.

Balancing Current and Future Needs

There is a growing emphasis on reuse of existing software and components in ways not planned for by the original designers, and this trend is expected to increase as technology becomes more pervasive. Inconsistencies between designed and future use provide further opportunities for security problems.

Requirements

The business requirements are often poorly defined for functionality as well as security. The multiplicity of factors generates diverse requirements and frequently leads to inherently conflicting ones. There may be requirements that are unknowable, such as when we do not fully understand the liability associated with new business activities. As system connectivity moves beyond the organizational boundaries, we have less knowledge of the external dependencies that exist. We often have to refine the requirements during development to reflect the knowledge gained. If non-functional requirements such as security are not specifically reviewed at critical junctions within the development life cycle, the opportunities for security problems from missing or incomplete requirements is extremely high.

Concurrent Evolution of Business Practices and Systems

One of the most significant characteristics of the business factors is that they can be very fluid as businesses respond to market changes and growing regulatory demands. New usage can generate new liabilities or raise the level of software assurance that must be demonstrated. The volatility of the business requirements compounds the costs and schedule slippages associated with the development of

large business systems. The tradeoff decisions must accommodate the variability driven by distributed management and business change. Software is always touted for its flexibility in terms of meeting requirements, but that flexibility is fully available only at the start of development. Design choices to meet specific requirements can constrain other options and limit the ability to make changes after the system is deployed.

Operations

There is a highly visible disconnect between the organizational visions of technology security and the realities of the implementations. One author describes the situation as "building a house so fragile that knocking on the door causes it to fall down—and then arresting the visitor who had the impertinence to knock" [Smith 05]. For example, students at a Pennsylvania high school who bypassed web access restrictions built into an application were charged with criminal trespass for using administrative passwords taped to the bottom of their loaner laptops [Kantor 05]. In another example, PharmaCare, a health insurer for Harvard University, provided access to pharmaceutical records based on birth date and student ID, which are public data [Russell 05].

Operations are increasingly non-stop. Changes cannot require system restarts and have to be made in minutes or hours and not weeks to respond to an operational condition. Diversity is a given. A small business may need to adapt its system to work with multiple large customers. A large organization may need to support outsourcing of services as well as joint business activities. The continuing evolution of usage and technology can rapidly age an application's architecture by invalidating design assumptions. Grady Booch, a co-founder of Rational Systems and now an IBM fellow, noted that many of the architectures he was assembling for his Architecture handbook had short "half-lives," say three to five years [Booch 06]. We may have that rapid architectural aging because the architectures were poorly designed, but we may also be seeing the effects of the changing operational requirements.

The operational system is also changing independent of development activities. Hardware and operating system upgrades are continuous as older versions fall beyond vendor support. Vulnerability monitoring and incident mitigation will introduce changes to infrastructure configurations and components such as fire walls and routers. The changing operational environment is a motivator for developing systems and software that is increasingly environment independent, further limiting the applicability of current accreditation, certification, and vulnerability analysis techniques.

Development - Architecture

The multitude of factors is a source of complexity for development. Booch made the following comment on March 22, 2005, in his web log [Booch 05]:

> *Most enterprise systems are architecturally very simple yet quite complex in manifestation: simple because most of the relevant architectural patterns have been refined over decades of use in many tens of thousands of systems and then codified in middleware; complex because of the plethora of details regarding vocabulary, rules, and non-functional requirements such as performance and security. Systems of other genres (such as artificial intelligence) are often far more complex architecturally.*

Complexity arises from the interaction of the non-functional requirements such as maintainability, performance, and security. For example, consider the effects of regulations and distributed systems on the functional architecture that supports access to employee information in a human resources database. The functional architecture is a relatively simple query and display. The use of access rules to support an internal corporate policy is straightforward. When such employee information moves among internal corporate systems or is exported to other organizations such as an insurance provider, the access and usage policy that was straightforward to implement in a single application must now be maintained across multiple applications and organizations.

Exporting data may raise regulatory issues such as those for the privacy of health care information, and we need to export the access policy that must be enforced. Data crossing international boundaries may be subject to additional regulatory constraints. For example, European Union privacy controls require individuals to allow their information to be shared (opt-in), whereas most U.S. organizations require individuals to decline sharing (opt-out).

Designs that provide greater flexibility are increasingly complex operationally. A legacy system might have used a static access policy and implemented that policy in application coding. On the other hand, reuse of a component in multiple contexts with differing access policies might lead to an implementation where the data provides a link to the policy represented in a manner that can be interpreted by the component. A simple binding of access decisions at compile time for the legacy system has been replaced by a more complex dynamic binding of the access policy at runtime. This dynamic solution provides broad flexibility. However, problem identification and correction can be extremely difficult unless capabilities to track and monitor the late binding decisions were part of the implemented solution.

There can be significant differences in how the system quality attributes are addressed. The analysis for hardware reliability may be based on well-established failure rates. We may be able to model user behavior with respect to various work processes to generate authentication and authorization requirements, but security also has to model an active agent, i.e. the attacker. Attackers do not have to respect a model.

A buffer-overflow exploit is a good example of the complexity of the security analysis in terms of the interaction of models. The attacker exploits a fault in a functional component. When that attack overwrites the call stack, the transitions between states are changed. Whereas the architect may have modeled the access control and authentication mechanisms and demonstrated that they satisfy the authorization requirements, the exploit enables the attacker to move outside of the implemented software controls and hence outside the model. The validity of the authorization model is now dependent on a security analysis of the data flow model. Social engineering exploits are also examples of external events that put a system in a state that may not be accounted for by initial analysis.

Distributed Systems

The distributed aspects of a business transaction also affect how we manage the interaction among systems. An asynchronous interface between systems may be used to avoid tying up computing resources waiting on a response from a system. With an asynchronous interface, a business purchasing transaction might start with a message from the purchaser to the supplier that describes the details of an order. An acknowledgement or a shipping notice would be a message from the supplier to the purchaser. Each message updates the transaction state maintained independently by both organizations. Conceptually, messages can be thought of as events, and the application architecture that processes events thought of as an event-driven architecture.

An event-driven architecture changes how we do authorization and authentication compared to the mechanisms that are used with a synchronous interface associated say with an interactive application. An interactive application could obtain the authentication information for a user and enforce the authorization policies whenever data is accessed. For 3-tier architectures (web client, server business logic, database server) the authorization and authentication is frequently done in the middle tier. For an event driven architecture, the system that processes a message cannot directly authenticate the submitter or verify that the submitter is authorized by the purchasing organization to submit the order. The submitter does not necessarily have control over the access of that information on the supplier's system and might want to be able to verify that the order had been officially accepted by the supplier. If we are using web service protocols, we

could incorporate some of the authentication and processing rules in the message. The message not only contains the data but also provide controls over access. Encryption might be used by the sender to restrict access to the information. Signing could be used to identify the authorizing agent of an order. The message can contain information that describes how that authentication was done, which could be used by the supplier as one criterion for accepting a transaction or as a mechanism that inhibits the purchaser from trying to deny that the order was authorized. This design approach is based on the assumption that operational infrastructure for all participants in the business process supports the needed encryption capabilities and that signatures are effectively established and maintained in such a way that they can be validated at each step of the process.

As we factor the business context onto distributed systems, we may move into uncharted territory such as dealing with the privacy of personnel information as data is shared among multiple systems and organizations. Host-based authorization and authentication have evolved into identify management, as we use standards such as the Security Assertion Markup Language (SAML) to share user identifiers and attributes across systems with independently managed security policies. The collection of security protocols for web services would add another level of architectural complexity.

Adaptability and Reuse

IT applications are often large monolithic structures, "one-off" designs that meet specific sets of requirements. The size and one-off nature of such systems can lead to higher costs, longer development times, and difficulties in modifying such systems to reflect changes in business processes. There is a strong motivation to consider a simple and easily tailored computer-supported service for multiple business processes in order to lower maintenance costs as the business processes evolve. The adjective agile is frequently applied, and the "IT bottleneck" is a popular target for complaints.

The objective to have software assembly correspond to mechanical assembly goes back to the beginnings of software engineering. Doug McIlroy, at the 1968 NATO Conference on Software Engineering, expressed that sentiment in a talk on "'Mass Produced Software Components":

> *Software components (routines), to be widely applicable to different machines and users, should be available in families arranged according to precision, robustness, generality and timespace performance. Existing sources of components—manufacturers, software houses, users' groups and algorithm collections—lack the breadth of interest or coherence of purpose to assemble more than one or two members of such families, yet software production in the large would be enor-*

mously helped by the availability of spectra of high quality routines, quite as mechanical design is abetted by the existence of families of structural shapes, screws or resistors [McIlroy 68].

We are now in the midst of another attempt to support reuse and improve assembly and integration, this time exploiting the advantages of service-oriented architecture (SOA) and web services. The web has demonstrated the effectiveness of a loosely coupled architecture for improved interoperability across diverse platforms. Web services can be used to implement an SOA. In an SOA, independent business services are built that can be easily composed and possibly even automatically assembled into a system to support a work process.

A skeptic might have noted the similarity of ideal rendition of SOA vision with the Lego-block analogy for component assembly. Grady Booch, in a November 15, 2004, entry in his web log, raises some of the concerns:

Service-oriented architectures (SOA) are on the mind of all such enterprises—and rightly so—for services do offer a mechanism for transcending the multiplatform, multilingual, multisemantic underpinnings of most enterprises, which typically have grown organically and opportunistically over the years. That being said, I need to voice the dark side of SOA, the same things I've told these and other customers. First, services are just a mechanism, a specific mechanism for allowing communication across standard Web protocols. As such, the best service-oriented architectures seem to come from good component-oriented architectures, meaning that the mere imposition of services does not an architecture make. Second, services are a useful but insufficient mechanism for interconnection among systems of systems. It's a gross simplification, but services are most applicable to large grained/low frequency interactions, and one typically needs other mechanisms for fine-grained/high frequency flows. It's also the case that many legacy—sorry, heritage—systems are not already Web-centric, and thus using a services mechanism which assumes Web-centric transport introduces an impedence mismatch. Third, simply defining services is only one part of establishing a unified architecture: one also needs shared semantics of messages and behavioral patterns for common synchronous and asynchronous messaging across services.

In short, SOA is just one part of establishing an enterprise architecture, and those organizations who think that imposing an SOA alone will bring order out of chaos are sadly misguided. As I've said many times before and will say again, solid software engineering practices never go out of style (crisp abstractions, clear separation of concerns, balanced distribution of responsibilities) and while SOA supports such practices, SOA is not a sufficient architectural practice [Booch 05].

The remainder of this article considers how separation of concerns and distribution of responsibilities can help manage complexity.

MANAGING COMPLEXITY

The articles in this content area focus on the complexity of system and organizational issues, but the components that make up those systems are still critical. The observation of one financial organization is that the resiliency of their systems in terms of threats, vulnerabilities, unexpected events, and change of practice had to be built in from the ground up. Complexity management will be impossible if we do not have predictable component behavior and if we have not incorporated the mechanisms to deal with the unexpected. The Scale: System Development Challenges article discusses the security challenges that arise as we increase the scale and scope of systems.

A range of analysis and mitigation approaches are under development to frame the complexity issues and aid in defining consolidated views to provide insight into potential gaps. As their value and scope are clarified, additional articles will be added to share successful mechanisms.

Start with the Challenging Problems

These articles identify more problems than solutions. The requirements are stretching the available techniques. The article by Praxis on Correctness by Construction (CbyC) provides appropriate top-level guidance for a slightly different context:

> *There is a natural tendency, when faced with a complex task, to start with the parts you understand in the hope that the less obvious parts will become clearer with time. CbyC consciously reverses this. Since risk and potential bugs hide in those areas that are complex and least-well understood those are precisely the areas that should be tackled first. Another reason for tackling uncertainty early is that freedom for maneuver tends to decrease as the project progresses; we don't want to have to tackle the hardest part of a problem at the point where we have the smallest range of design options open to us. Of course, we could take the fashionable approach and refactor our design; however, designing, building and incrementally validating a system only to change it because we failed to consider risky areas early enough is hardly efficient and is manifestly not correctness by construction!*
>
> *The High Level Design describes the architecture of the software. This is where we ensure that key non-functional properties such as safety and security are addressed. It is also the point where we make provi-*

sion for unresolvable requirements uncertainties by selecting a design that is flexible in the areas where change is probable. Rather unintuitively, CbyC's response to requirements uncertainty is more design not less!

Security practitioners have always complained that security is addressed too late. The increasing complexity and number of interdependencies may make it very difficult if not impossible to reengineer security into an established architecture.

Identify What Is Secure Enough

Evidence of the development challenges appear early in the software development life cycle. The requirements are typically incomplete and fluid and may need to accommodate extensive variability because of stakeholder conflicts that cannot be resolved. Tradeoff decisions must accommodate a range of variability driven by distributed management, business change, and infrastructure change. The diversity of usage, available software, operations, risks, and organizational risk tolerance leads to unique characteristics for each organization. Each organization has to answer the question as to what is secure enough for their usage and risks. The expanded scope and scale increase the risks for organizations that do not clearly articulate their security needs in requirements and for development.

Regulations are often an organizational driver for security and an important parameter in deciding what is secure enough. Regulations such as Sarbanes-Oxley have prompted organizations to implement tighter governance polices for financial systems. The demands of HIPAA (Health Insurance Portability and Accountability Act) compliance have motivated organizations to change the usage of data such as social security numbers. Software provides both control and auditing capabilities.

Commercial web sites have raised new liabilities. In 2003, a SQL injection flaw discovered in the PETCO commercial website exposed up to 500,000 credit cards to outside access. There was no evidence that the vulnerability had been exploited, but a consequence of the flaw was increased operational and development expenses for the next 20 years. An FTC investigation ensued on alleged deceptive trade practices, as PETCO's privacy statement included the phrase "At PETCO.com, protecting your information is our number one priority, and your personal information is strictly shielded from unauthorized access." PETCO agreed to a 20-year settlement with the FTC in which PETCO is prohibited from misrepresenting the extent to which it protects the security of customers' personal information and must establish and maintain a comprehensive information security program that is certified by an independent professional every two years for the 20-year life of the order.

Balance Consolidation of Services, Separation of Concerns, and Delegation of Responsibilities

Separation of Concerns

The ways that we addressed complexity in a small or relatively simple system may not be valid as we expand the scope and scale. Separation of concerns and consolidation are two classic techniques for managing complexity. A motivation for separation of concerns is to decompose the system into more manageable and understandable parts to be able to encapsulate and manipulate the parts that correspond to a particular concern. For example, classes in object-oriented development represent a separation based on data concerns, and such a separation facilitates the relatively independent development of the functions. A software architect would like to maintain separation of concerns for essential system qualities such as performance and security so that components that significantly affect those qualities are encapsulated in such as way that performance and security issues can be addressed relatively independently. Consolidation seeks commonality. Can a software function or service be shared? Service-oriented architectures are one example of consolidation. However, the expanding scope and scale challenge how we have previously decomposed systems or consolidated functions.

Delegation of Responsibilities

An essential security task is delegating the responsibility for meeting the requirements. Whereas separation of concerns has been primarily a software development technique, an analysis of the delegation of responsibilities includes software, hardware, users, and system management. For example, authentication responsibilities are shared by users and the system in that a password or private key used for authentication can be compromised by a careless user. The user responsibilities might be reduced by using a one-time password mechanism or a biometric device such as finger-print scanner.

The delegation of responsibilities can purposely introduce redundancy to support a defense-in-depth strategy. A simple form of defense in depth is to always check the validity of inputs to a component even though the design calls for those checks to be made in advance of the call to the component. An objective for defense in depth is to avoid a single control point that might be compromised by an attacker.

A poor delegation of responsibilities is often reflected by an “It’s not my job” response and inaction when problems arise. We usually associate that response with operations staff, but it equally applies to those involved with system development. The causal analysis for engineered systems usually concentrates on component failures that are mitigated by prevention or redundancy. That focus

does not account for (1) social and organizational factors in accidents, (2) system accidents and software errors, (3) human error, and (4) adaptation over time [Leveson 05]. It is difficult to identify a single root cause for the 2003 power blackout that is described in [US-Canada 04]. A risk for security is that it is typically treated as a separate concern, with responsibility assigned to different parts of the organization that often function independently, and that isolation is even more problematic as the scope and scale of systems expand. The power blackout article suggests some first steps to take to better manage failures.

Business integration requirements and the appearance of technologies such as web services to support that integration of distributed systems can affect the delegation of responsibilities. It is now not unusual to find that an organization's development, operational, and business groups are tackling common problems with little coordination or that some security problems have been ignored.

Consolidation of Services

The multiplicity of systems and increasing number of possible error states arising from the interactions can overwhelm the analysis. The risk is having too many point solutions that mitigate narrowly specified events. Changes in usage could generate a significant reengineering effort. An argument can be made that we frequently have too much separation among system qualities such as security, reliability, and maintainability.

This section poses observations on current practice by organizations that have to find ways to better manage system complexity. Such practices are on the leading edge and certainly have not been proven.

As noted by the Burton Group in one of their client reports, a number of organizations are revisiting how they treat availability. Is it a security requirement or a business requirement? Should those two perspectives be consolidated? As a business requirement, availability supports business continuity, which is based on the dependability of the computing infrastructure, service providers, the technology deployed, operations, information processing and communications. Business continuity is primarily achieved with redundancy of equipment, personnel, and computing locations. Security, reliability, and compliance are all part of business continuity.

With respect to business continuity, the integration of geographically distributed business units could be implemented in ways that provide value in maintaining continuity of operations and assist in removing single points of failure, be they facilities, people, or processes.

- The design guidance for the enterprise architecture, for example, might be to

- disperse information technology processing so that there was no dependency on any one location
- disperse critical functions among multiple sites
- enable both near-site and far-site (possibly international) recovery
- enable a specific geographical region to operate independently
- design for flexibility
  - delay bindings of features that are likely to change to enable last-minute customization. For example, if a business function has to be moved to an international site, software changes may be required for compatibility with that function's operations.
  - emphasize commonality to support flexibility

Aspects of such guidance could be applied to enterprise architectures that are not geographically distributed. A tactic for supporting business continuity for a system of systems would be to maintain sufficient independence among the systems so that essential aspects of business processing can be restored with a subset of the systems rather than the full system of systems or that processing can continue asynchronously with an eventual synchronization. In this respect, various non-functional aspects such as adaptability can provide synergy in support of security based on how they are instantiated.

Integrate Operational and System Risk Analysis

An integrated risk assessment should identify the security risks that arise from a poor coordination of responsibilities between development and operations or among organization units. Organizations must recognize that each development effort drives change into the current operational environment that can create gaps with current operational policies and practices. By taking a proactive approach to identifying and evaluating the potential operational risk at critical points throughout the software development life cycle, an organization can have visibility into gaps and issues before crises occur. An approach to considering operational security from within the development life cycle prior to actual deployment is described in Considering Operational Security Risk During System Development. The application of this approach iteratively at appropriate milestones such as requirements review, design review, integration validation, and stakeholder acceptance can provide opportunities to identify and adjust development decisions and issues with operational policies and practices to mitigate unacceptable risk.

CONCLUSION

The expanding scale of business usage of information technology increases the importance of security analysis, considering the people and organizational issues in addition to the technology. The classic design techniques that have been used to manage complexity often need to be refined to reflect the complex operational environment. The article Scale: System Development Challenges provides a more detailed discussion of some of technical issues raised in this introduction.

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

DM-0001120