Chapter 6. Security and Complexity: System Assembly Challenges

6.1. Introduction

The primary theme of this chapter is how aspects of complexity due to technical difficulty, size, and conflicting objectives affect security as systems expand to support multiple processes within and across organizations.[1] Mitigation strategies and project management approaches are suggested for each area, including examples of "planning for failure" in the context of Web services and identity management.

System development has always encountered new and often complex problems that were not represented in project plans. Often, the hard-to-solve problems are not new. Not many years ago, for example, the Common Object Request Broker Architecture (CORBA) received considerable attention as an approach for integrating distributed systems; now much of that attention has shifted to Web services. The sequence of solutions over multiple decades is not just a measure of the difficulty of the challenges, but also an indicator of the progress that is made in overcoming them. For example, Web services protocols have significantly eased the mechanics of connecting systems and offer the opportunity to address a number of business requirements.

The improved capability to assemble systems does not by itself address the problem of the failures observed in complex systems, but rather may increase the risk of deploying systems whose behavior is not predictable. The theme of a 2007 *New York Times* article is captured in a quote by Peter Neumann: "We don't need hackers to break the systems because they're falling apart by themselves" **[Schwartz 2007]**. For example, 17,000 international travelers flying into Los Angeles International Airport were

stranded on planes for hours one day in August 2007 after U.S. Customs and Border Protection Agency computers went down and stayed down for nine hours. The Northeastern power grid failure in the summer of 2003 is another recent example of the effects of a system failure. Voting machine failures continue to be publicized. Customers for Skype, the Internet-based telephone company, encountered a 48-hour failure in August 2007.

The Los Angles airport failure was traced to a malfunctioning network card on a desktop computer that slowed the network and set off a domino effect of failures on the customs network. The power grid failure was not caused by a single event but rather by a cascading set of failures with multiple causes. Avi Rubin, a professor of computer science at Johns Hopkins University, noted that for voting machines the focus might have been too much on hackers and not on accidental events that sometimes can cause the worst problems. The Skype failure was initiated by a deluge of login attempts by computers that had restarted after downloading a security update. The logins overloaded the Skype network and revealed a bug in the Skype program that allocated computer resources that normally would have mitigated the excessive network load **[Schwartz 2007]**.

The individuals interviewed for the *New York Times* article include a number of well-known experts in computer security, but the general observations focused more on reliability and complexity than on security:

• Most of the problems we have today have nothing to do with malice. Things break. Complex systems break in complex ways. (Steve Bellovin, Columbia University)

• We have gone from fairly simple computing architectures to massively distributed, massively interconnected and interdependent networks. As a result, flaws have become increasingly hard to predict or spot. Simpler systems could be understood and their behavior characterized, but greater complexity brings unintended consequences. (Andreas M. Antonopoulos, Nemertes Research)

• Change is the fuel of business, but it also introduces complexity, whether by bringing together incompatible computer networks or simply by grow-

ing beyond the network's ability to keep up. (Andreas M. Antonopoulos)

• Complexity was at the heart of the problem at the Los Angeles airport. Problems are increasingly difficult to identify and correct as we move from stovepipes to interdependent systems. (Kenneth M. Ritchhart, the chief information officer for the U.S. Customs and Border Protection Agency)

This chapter considers how some of those observations relate to security analysis. One perspective is that insufficient effort has been made to analyze potential failures and to apply known methods and technology. If a system is designed correctly in the first place, it can be made reliable, secure, fault tolerant, and human safe. The focus of this perspective, then, is developing and using best practices and better integrating security into the overall SDLC. Security analysis should be able to take advantage of an increased emphasis on managing failures. Attackers create events that take advantage of errors that are ignored or poorly managed by the system. **Section 6.2** provides an introduction to categories of errors that are associated with security failures. Security analysis involves considering both functional and attacker perspectives of a system. In **Section 6.3**, those perspectives are applied to the examples of Web services and identity management.

Unfortunately, existing best practices do not fully address the security challenges caused by increased system complexity and distributed operational environments. The wider spectrum of failures, changing and conflicting goals, and incremental development challenge some of the traditional security assumptions. For instance, risk assessments are affected by reduced visibility for distributed systems and the wider spectrum of failures. Large systems are developed incrementally and must deal with changing and sometimes conflicting stakeholder objectives. As a consequence, security now needs to consider general system failures that usually have been associated with reliability. The security research world is constantly evolving, and assumptions about certain vulnerability classes can change overnight **[Howard 2007]**. Howard emphasizes a critical lesson that most vendors have learned the hard way: Today's denial of service is tomorrow's exploit. An attacker could, for example, exploit any of

a variety of failures **[Schwartz 2007]** to create a denial of service. The security challenges created by increased system complexity may prove to be difficult-to-mitigate problems that do not have known solutions. The effects of multisystem complexity on security are discussed in **Section 6.4**, and **Section 6.5** provides an overview of approaches for managing deep technical problems.

6.2. Security Failures

A failure—an externally observable event—occurs when a system does not deliver its expected service (as specified or desired). An error is an internal state that may lead to failure if the system does not handle the situation correctly; a fault is the cause of an error. For example, in a buffer overflow, the error might be in a functional component that does not check the size of user input. An attacker could exploit that error by sending an input stream that is larger than available storage and that includes executable code: This is the fault. The program logic that accepts the bad input actually overwrites another part of program logic. The attacker then executes the added code, which enables him or her to bypass authentication controls. In this way, the functional error can be leveraged into a security failure.

A number of errors have been demonstrated to lead to exploitable failures. Historically, a significant number of security vulnerabilities have been associated with errors in functional components rather than in security functions, such as those employed for authentication and authorization. An attacker may, for example, try to put a system into a state that was not anticipated during development. Such a situation might lead to a system crash and hence a denial of service, or it might let the attacker bypass the authentication and authorization controls and access normally protected information. Attacks have also exploited errors in parts of the system that were not fully analyzed during development or that were poorly configured during deployment because that effort concentrated on primary usage. Attacks can also create an unlikely collection of circumstances that were not considered in the design and exploit aspects of an interface that developers have left exposed.

Software engineers should be aware of the ever-lengthening list of exploitable errors (although even an expert can find it difficult to navigate through all of them). Specific weaknesses or underlying issues that that may cause vulnerabilities are described in the Common Weakness Enumeration **[CWE 2007]**. Several systematic, in-depth, and lengthy discussions have examined the difficulties associated with building trustworthy systems **[Anderson 2001**; **Neumann 2004**; **Schneider 1999]** and can provide a solid foundation of awareness. Although these sources concentrate on software and system errors, be aware that a design may also need to mitigate exploitable operator and user errors.

6.2.1. Categories of Errors

To aid in the analysis of security failures, errors can be categorized according to their occurrence in these five system elements:

1. *Specific interface.* An interface controls access to a service or component. Interfaces that fail to validate input often crop up on published vulnerability lists.

2. *Component-specific integration.* Assembly problems often arise because of conflicts in the design assumptions for the components. Project constraints may require using components, COTS software, or legacy systems that were not designed for the proposed usage, which raises the likelihood of mismatches. The increasing importance of business integration requirements compounds the problems associated with component integration and serves as the motivation for designs based on a service-oriented architecture.

3. *Architecture integration mechanisms.* Commercial software tool vendors often provide a built-in capability for purchasers to integrate a tool into their systems and tailor its functionality for their specific needs. Unfortunately, the capability to reconfigure a system rapidly is typically accompanied by an increased probability of component inconsistencies generated by the more frequently changing component base, as well as the increased risk that the dynamic integration mechanisms could be misused or exploited. These mechanisms represent another interface that must be properly constrained **[Hoglund 2004]**.

4. *System behavior: component interactions.* The behavior of a system is not the simple sum of the behaviors of its individual components. System behavior is strongly influenced by the interactions of its components. Components may individually meet all specifications, but when they are aggregated into a system, the unanticipated interactions among components can lead to unacceptable system behavior. Components that are not secure as stand-alone components in an operating environment may be secure when used in a system that controls access to those components. The technical problems for this category of errors can be significantly more challenging to solve than the corresponding problems for the first three categories.

5. *Operations and usage.* Operational errors are also a frequent source of system failures, as noted in *Trust in Cyberspace* **[Schneider 1999]**:

> Errors made in the operation of a system also can lead to system-wide disruption. NISs are complex, and human operators err: An operator installing a corrupted top-level domain name server (DNS) database at Network Solutions effectively wiped out access to roughly a million sites on the Internet in July 1997 **[Wayner 1997]**; an employee's uploading of an incorrect set of translations into a Signaling System 7 (SS7) processor led to a 90-minute network outage for AT&T toll-free telephone service in September 1997 **[Perillo 1997]**. Automating the human operator's job is not necessarily a solution, for it simply exchanges one vulnerability (human operator error) for another (design and implementation errors in the control automation).

6.2.2. Attacker Behavior

Modeling attacker behavior presents significant challenges in software failure analysis. The analysis for quality attributes such as performance and hardware reliability is based on well-established failure rates, but security analysis does not have as solid a foundation. We may be able to model the work processes so as to generate authentication and authorization requirements, but we also have to model an active agent—the attacker—who can change the details of an attack in response to defensive actions.

The buffer overflow exploit described earlier in this chapter is a good example of the complexity of security analysis in terms of the interaction of models. Whereas the architect may have modeled the authentication and authorization mechanisms and demonstrated that they satisfy the design requirements, the new code that the exploit allowed to be executed enables the attacker to move outside the implemented software controls and hence outside the model. As a consequence, the validity of the authorization model becomes dependent on a security analysis of the data flow.

Social engineering exploits are other examples of external events that put a system in a state that may not be accounted for by the usage models. In these exploits, an attacker typically tries to convince users or administrators to take an action that lets the attacker circumvent a security control. For example, the attacker might try to impersonate a user and convince help-desk personnel that they should change an account password so that a customer order could be submitted.

Attack patterns, which are discussed in **Chapter 2**, are a good way to describe known attacker perspectives of a system.

6.3. Functional and Attacker Perspectives for Security Analysis: Two Examples

Security analysis must take both a functional perspective and the attacker's perspective. The functional perspective identifies the importance of an issue to the business functionality of the system and hence is a component of a risk assessment. The attacker's perspective considers the opportunities that business usage and the specifics of a technology create. For example, a capability to easily configure Web services could also be used by attackers to configure those services to support their objectives. Central services that consolidate authentication and authorization are also highly prized targets for attackers, because any vulnerabilities in those services can provide access to desired targets such as business information assets.

These two perspectives capture the fact that distributed decision making across both physical and organizational boundaries is a necessity for software-intensive systems that support human interactions. As work processes extend beyond the corporate IT perimeter and encompass services and data provided by external systems and organizations, the concept of a perimeter becomes even more elusive. Frequently each interface must be monitored to reflect the dynamically changing assurance associated with it. The interfaces among systems may depend on organizational relationships. Thus the central control represented by a firewall-protected perimeter has increasingly been replaced by multiple, and potentially conflicting, control points.

The next two sections introduce some of the security issues associated with Web services and identity management. Web services are often deployed to support business requirements for the integration of geographically distributed systems. Identity management concentrates on one aspect of system interoperability: authentication and authorization across multiple systems. We take both a functional perspective and attacker's perspective with each example.

By "functional," we mean the added value from the organizational perspective. For Web services, the functional perspective includes the capability of dynamically exchanging information without having to hardwire the mechanics of that exchange into each system. Thus a change in business requirements may be implemented by changing the Web service interfaces rather than changing the functional business logic. For identity management, the functional perspective is represented by two objectives:

• Interoperability across systems to enable sharing of identity information

• Consolidation and, where desired, centralization of the security services of authentication and authorization across multiple systems

6.3.1. Web Services: Functional Perspective

To support greater business efficiency and agility, information systems and their operations have become increasingly decentralized and hetero-

geneous. Business processes are distributed among far-flung business divisions, suppliers, partners, and customers, with each participant having its own special needs for technology and automation. As a consequence, the demand for a high degree of interoperability among disparate information systems has never been greater. Moreover, this high degree of interoperability must be sustained as participants continually modify their systems in response to new or changing business requirements.

Traditional assembly and integration methods (and the resulting integration software market stimulated by these methods) are not particularly well suited to this new business environment. These methods rely on a tight coupling between cooperating systems, which requires either the universal deployment of homogeneous systems (unlikely, considering the diversity and broad scale of modern business services) or extraordinarily close coordination among participating development organizations during initial development and sustainment (e.g., to ensure that any changes to APIs or protocols are simultaneously reflected in all of the deployed systems). Such tight coordination is often impractical (e.g., prohibitively expensive), and rapid evolution in response to a new business opportunity is typically out of the question.

In contrast to traditional assembly and integration methods, Web services technology uses messages (in the form of XML documents) that are passed among diverse, loosely coupled systems as the focal point for integration. These systems are no longer viewed solely as components within a larger system of systems, but rather as providers of services that are applied to the messages. Web services are a special case of the more general notion of service-oriented architecture (SOA). Service-oriented architectures represent interconnected systems or components as collections of cooperating services. The goal of Web services technology is to dramatically reduce the interoperability issues that would otherwise arise when integrating disparate systems using traditional means.

The distributed aspects of a business transaction also affect how we manage identities. A Web services message, for example, might contain an order for materials and be submitted by an organization to a supplier. For infrequent transactions, an employee from the purchasing organization

could log into a supplier's online application and submit the request. That employee would be authenticated by the supplier's identity management system. This type of online purchasing system is synchronous—that is, the supplier's system requests information and waits for a response. Such a synchronous communication protocol can tie up computing resources waiting for responses. In contrast, for high-volume transactions (and particularly those associated with just-in-time delivery), the transactions are system generated and the communication protocols are asynchronous—that is, the sending system does *not* wait for a reply. With an asynchronous interface, a business purchasing transaction might start with a message from the purchaser to the supplier that describes the details of the order. Later, the supplier would send another message—an acknowledgment of the order or confirmation of shipment—to the purchaser. Each of these messages updates the transaction state that is maintained independently by both organizations. In such a communication process, messages can be thought of as events, and an application architecture that processes messages is often described as an event-driven architecture.

In an event-driven architecture, the system that processes a message needs the equivalent to the user login for the synchronous online application to authenticate the purchaser and, in some instances, to verify that the purchaser is authorized by the purchasing organization to submit the order. The purchaser does not directly log into the supplier's system; rather, the purchaser's user authentication and authorization information is incorporated into the order sent to the supplier. Although this purchase order may contain business-sensitive information that should require authentication and authorization to access, the purchaser cannot directly observe or monitor the processing of that order by the supplier. As a consequence, the purchasing organization may require assurance that the order was accepted by the supplier.

A Web services message not only contains the ordering data, but also provides mechanisms for the necessary authentication and authorization. Encryption might be used by the sender to restrict access to this information. Signing can be used to confirm data integrity or as a means to identify the author of specific components of the message. Security Assertion

Markup Language (SAML) can be used to share user identities and attributes.

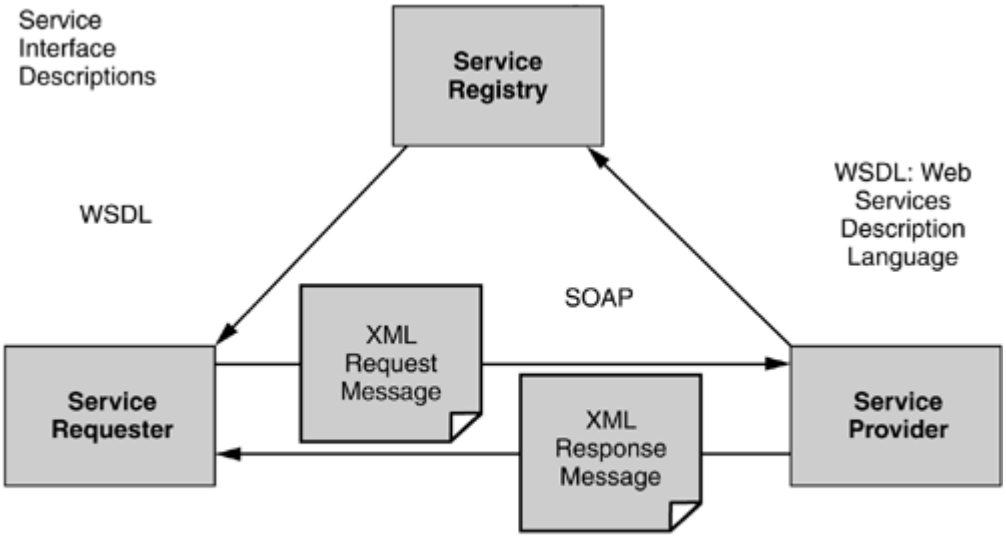
6.3.2. Web Services: Attacker's Perspective

We next look at the attack risks associated with Web services. Our analysis assumes that Web services have been implemented using the Simple Object Access Protocol (SOAP). SOAP is an XML-based protocol that lets applications exchange information. The information exchanged might consist of business data (such as a purchasing order) or instructions on how the business process should be done.

This section describes illustrative—not canonical—threats and vulnerabilities that Web services applications face. To do so, it uses Shirey's model **[Shirey 1994]**, which categorizes threats in terms of their impact as disclosure, deception, disruption, and usurpation. The threats are further categorized by the service-level threats that are common to most distributed systems and the message-level threats that affect Web services XML messages. A more detailed description of the threats and mitigations appears in **[BSI 27]**.

Figure 6–1 depicts an exchange based on Web services. Web services are designed to support interchanges among diverse systems. The initial step of an interchange is for the purchasing organization to acquire a description of the data transaction and the protocols used for encryption and signing. The eventual objective is for the initial exchanges to establish security policies that are acceptable to both parties. A service registry contains the message exchange pattern, types, values, methods, and parameters that are available to the service requester. It could be controlled by the service provider or could be a more widely accessible site operated by a third party.

Figure 6-1. *Web services*



Two main risk factors are associated with Web services:

• *Distributed systems risks*— that is, risks to the service providers themselves that are similar to risks that exist in Web applications and component applications. For example, malicious input attacks such as SQL injection fit this description. These risks arise simply because the system is distributed on a network. Note that standard IT security controls such as network firewalls are largely blind to Web services risks, because Web services are deployed on commonly available open ports. Nevertheless, some types of application firewalls have the ability to examine content, such as XML message bodies, and can use application-specific knowledge to thwart some attacks. Unfortunately, they are by no means a panacea for all distributed systems risks **[BSI 28]**.

• *Message risks*— that is, risks to the document and data that are exchanged among participants. The document may participate in a multisystem transaction or be subject to inspection by a variety of intermediaries, each operating in different security zones, including separate policy, geographic, technical, and organizational domains. The message's content may also, of course, contain sensitive data.

Using Shirey's threat categories based on impact, **Tables 6–1** through **6–4** describe general threats for Web services, the tactics that an attacker might use, and the ways that a developer might mitigate those threats.

Studying attack patterns that use those tactics can assist in analyzing the possible mitigation options.

Table 6-1. *Disclosure of Information*

Attack tactics	The XML-based messages may be passed without encryption (in the clear) and may contain valuable business information, but an attacker may be more interested in gaining knowledge about the system to craft attacks against the service directly and the system in general. A security vulnerability in the service registry (like that shown in Figure 6–1) might let the attacker identify the system's data types and operations. That same information could be extracted from messages sent in the clear. Messages may also contain valuable information such as audit logs and may lead to identity spoofing and replay attacks, which use message contents to create a new message that might be accepted.
Mitigations	Authentication and authorization mechanisms may be used to control access to the registry. There are no centralized access control mechanisms that can protect the XML messages, but message-level mechanisms such as encryption and digital signatures can be used.

Table 6-2. *Deception*

Attack tactics	An attack can try to spoof the identity of the service requester by sending a well-formed message to the service provider. The identity of the service provider could also be spoofed. XML messages are passed without integrity protection by default. Without integrity protection, an attacker could tamper with the XML message to execute code or gain privileges and information on service requesters and providers.
Mitigations	Web services provide a number of integrity and authentication mechanisms that can mitigate deception. For example, WS-Security defines how to include X.509, Kerberos, and username and password security information in the XML message to support end-to-end authentication. Message integrity is supported through digital signatures and message origin authentication.

Table 6-3. *Disruption*

Attack tactics	An attacker could execute a denial of service at the network level against a Web service. Messages could also be used for a denial-of-service attack. For example, an attacker could send a specially formed XML message that forces the application into an infinite loop that consumes all available computing resources. The receipt of a large volume of malformed XML messages may exceed logging capabilities.
Mitigations	A network-level denial of service is mitigated in a similar fashion to a Web application denial of service—that is, by using routers, bandwidth monitoring, and other hardware to identify and protect against service disruption. Mitigation of message-level disruptions depends on validating the messages, but that mitigation can be tricky, because the target of such attacks is that mitigation component. One tactic would be to encapsulate message validation in a service that is applied before messages are passed to applications.

Table 6-4. *Usurpation*

Attack tactics	An attacker may usurp command of a system by elevating his or her privileges. One way to do so is to exploit the service registry to redirect service requests, change security policy, and perform other privileged operations. XML messages may be used to propagate viruses that contain malicious code to steal data, usurp privileges, drop and alter tables, edit user privileges, and alter schema information.
Mitigations	When service registries are used in Web services, they become a central organizing point for a large amount of sensitive information about services. The service registry (and communication to and from the service registry) should be hardened to the highest degree of assurance that is feasible in the system. Vulnerability analysis of source code pays particular attention to system calls to privileged modules in the operating system. The service registry can affect policy, runtime, and locale for other services and hence is analogous in importance to the operating system. Therefore particular attention must be paid to how service requesters access the service registry. At the message level, vendors are beginning to realize the significant threat that viruses, when attached and posted with XML documents, may pose to the environment. For systems that may have XML or binary attachments, virus protection services should be deployed to scan XML and binary messages for viruses in a similar fashion to email messages—that is, before the messages are executed for normal business operations.

6.3.3. Identity Management: Functional Perspective[2]

Information systems are increasingly interconnected, such as when companies' intranet sites provide single sign-on capabilities to other companies that provide 401(k) and health benefit services. These parties—that is, the financial and health benefits services organizations—may rely on

their customers' systems to provide information about the identities of the users who are connecting to their services. However, the two systems may not have consistent security policy, enforcement, audit, or privacy requirements.

Identity management (IM) is an administrative system that deals with the creation, maintenance, and use of digital identities. The use of a digital identity is controlled by the authorizations associated with it. IM includes the business processes associated with organization governance, as well as the supporting computing infrastructure. Operating systems, specific applications, and database management systems have each defined their own digital identity and implemented access control mechanisms. The initial technical challenge associated with implementing organizational governance policies is whether those policies can be implemented consistently across that diverse collection of identities and access control mechanisms. Even more difficult technical challenges arise from the following trends:

• Identity must be consistently managed across multiple and often geographically distributed systems. In some instances, identities can pass between organizations. A number of technical issues have to be resolved before such management is possible, however:

– **Access control.** Identity is a foundation-level component for many access control mechanisms. Identity information about a digital subject is bound to a principal, which is typically an end user. Access control mechanisms consume identity data from the principal to make and enforce access control decisions. Weaknesses in identity systems affect the overall viability of access control, security, and privacy mechanisms.

– **Audit and reporting.** These systems can be used to record, track, and trace identity information throughout systems. Audit logs and usage reports may be used for regulatory, compliance, and security purposes. Depending on their implementation, however, they may create privacy issues for individuals when that information is reported. Some techniques allow for system reporting and monitoring without disclosing identity information.

– **Identity mapping services.** Distributed systems may have different implementations of identities. Identity mapping services can transform identities in a variety of ways so that a principal on one system can be mapped to a principal on another system.

– **Domain provisioning services.** In this context, a domain is a system in which computing subsystems have common definitions for security objects such as identities and authorizations. A domain could, for example, consist of a collection of Windows workstations or be a database management system deployed on a single server. The organizational identity and authorization information must be mapped to each domain. Provisioning services perform that function.

• Identities are associated with more than just employees. Some organizations may have to manage customer, contractor, and business partner identities as well.

• Increasingly, legislation and regulation have begun to recognize the value of identity data. Countries and industries have specific points that must be addressed to ensure that identity is protected. For applications that have an international user base, additional regulatory and legal concerns may span legal boundaries.

• Privacy concerns relate to identity information that is linked at some level to an individual. They center on which personal data is disclosed and may manifest themselves in the system design through privacy legislation, liability, and/or psychological acceptability and success of the solution. Systems may implement privacy mechanisms using pseudonyms or anonymous mechanisms.

• Information relating to digital subjects is used by a wide array of applications from Internet portals (e.g., business Web sites, loyalty programs, customer relationship management services, personalization engines, and content management servers) to enhance the customer experience and provide convenience and targeted services on behalf of businesses and consumers. Personal data, when stored by organizations, may also be shared and correlated for a variety of reasons, including data mining and

target marketing; these uses of personal data may directly conflict with goals related to pseudonymous protection of data subject information.

6.3.4. Identity Management: Attacker's Perspective

Given that identity information is so central to so many security decisions and to so much application functionality, it represents a highly prized target for attackers. From a cultural viewpoint, identity information is understood to require extra due diligence by government, regulatory bodies, and individual users. This kind of information and its related architectural constituents, therefore, may be held to a higher standard for both security and privacy elements, and additional security analysis, design, implementation, operations, and auditing may be required. Throughout all phases of the SDLC, you should examine the security model of the identity services and identity stores in the context of your overall system security to ensure that those services and stores are among the strongest links in the system. The more identity information is centralized logically or physically, the more risk to identity information is aggregated.

Security Versus Privacy

An inherent tension exists between security and privacy that plays out most directly in the identity space. This tension revolves around the extent to which the user and the relying party have control and visibility of personal data. To be effective, the identity architecture must resolve these concerns in a manner that is congruent with each party's requirements.

Identity information leakage can occur when identity providers supply more information than is necessary to perform the functional task and do not protect the identity information when it is transmitted across the domains' boundaries. A classic example would be a service that requires authorized users to be 21 years of age or older. In this case, the relying party asks the identity provider for the age information. If the identity provider gives the relying party the user's birth date so that the relying party can calculate the age of the user, then the user's birth date has been propa-

gated to a separate service that now can retain (or disclose or otherwise lose) a valuable piece of personal information that the service does not absolutely require to perform its functions. A more appropriate response could be that the relying party queries the identity provider or the data subject if the user is more than 21 years old and receives a Boolean yes/no response. Some information has been revealed to the service provider in this instance, but it is far less critical.

Emerging technologies such as Web services and federated identity have direct implications on identity information leakage. An objective for federated identity is to enable a portable identity by sharing identity information among normally autonomous security domains. With federated identity, a traveler could, for example, log into a hotel system and then reserve a rental car or reconfirm an air reservation without explicitly logging into the car rental or airline systems. Those systems would accept the user's identity as authenticated by the hotel. Early efforts related to portable identity for Web usage, such as Microsoft Passport, suffered from disclosure of identity information to parties that did not have a justifiable place in the transaction **[Cameron 2005]**. Directory services that replicate identity information at the data level can also create exposure by replicating more identity information than is required for dependent systems.

General mitigations for identity management risks are listed in **Table 6–5**.

Table 6-5. *Identity Management Risk Mitigations*

Availability	Identity services provide an interface to information about subjects stored in the identity stores in a system. They also can provide a single point of failure that attackers may target to bring application systems down, without the need for the attackers to target the application itself. In fact, because identity services and stores are often reused in organizations serving identity information to multiple applications, an attacker who successfully executes a denial-of-service or other availability attack against identity services and stores can have a large adverse impact on the availability of the system. Incorporating redundancy and automatic failover for identity services can be used to combat availability threats. Services are often consolidated to reduce costs, but consolidation of identity services can expand the consequences of a successful exploit. Decentralizing the deployment and management of identity services may be an appropriate tradeoff for this risk, albeit with increased operational costs.
Hardened servers and services	Given the critical nature of the data that identity servers host and the access they vouch for in the system, identity servers should be hardened to the highest level of surety that is practical. The goal of identity servers to is provide and verify identity information for applications—not to run Web servers, database servers, and so on. Standard server-hardening techniques that limit privileges and services available only to those strictly necessary apply in this instance. Hardening special-purpose identity servers such as directory services servers is a relatively more straightforward task than hardening identity servers; the latter are more general-purpose tools in the organization and may contain both identity and line of business or domain information. Host integrity monitoring, network- and host-based intrusion detection systems, network security monitoring, and secure exception management practices enable more robust detection when protection mechanisms fail.

Incident response	Many attacks against identity—and particularly identity theft attempts—rely in large part on the victim remaining ignorant that theft has occurred for some period of time. The damage an attacker can cause can be partially mitigated by an effective, rapid, and targeted response to identity data theft. An effective program could include clear communication lines and response patterns, along with a set of guidelines that the victimized users can implement to deal with the aftermath of an identity theft.
Usability: endpoint attacks	At runtime, the endpoint for identity data is frequently the user session and user desktop. Therefore, securing identity information often boils down to a battle between usability and security. The work done in protecting an identity across dozens of hops across servers and nodes can be defeated by attackers who target the desktop layer or the user. Robust identity systems must ensure that the usability of identity is factored in so that users understand their roles and responsibilities in using their identity in the system. Individual users are typically not able to discern when it is appropriate and safe to disclose personal information.

6.3.5. Identity Management and Software Development

Software development teams may lack agreed-upon plans for adoption of standard representation and consumption patterns for authentication, attribute query or update, and authorization of identity information across technological and organizational domains. The current state of identity may consist of numerous identity silos that are directly bound to domain-specific technologies, policies, and organizational domains, each with its own interpretation of how to issue, encapsulate, and negotiate identity data and services. This potential lack of consistency creates issues for distributed systems that are required to traverse multiple identity silos and domains and has the overall effect of stimulating numerous one-off solutions for identity, each of which contains its own arcane, tightly coupled, and technology-specific ways of dealing with identity. There is a well-understood best practice in software development that developers should not attempt to write their own cryptographic algorithms because of the complexity, lack of peer review, and value of that which the cryptographic functions are protecting. Developers, in contrast, routinely write one-off identity solutions that are never peer reviewed by a wider audience. This identity information is then propagated and integrated throughout software systems and used as a basis for making security decisions about access control to critical resources and the confidentiality of personal and business data.

In many systems, these one-off solutions are further integrated with other identity silos, creating a mishmash of identity solutions with varying limitations, and in the worst case generating a lowest common denominator effect. Exacerbating this problem further is the fact that many identity solutions are already in place as legacy systems while software is being developed, such that the projects inherit the standard issues found in legacy integrations with identity, including brittleness and lack of support for robust protocols and current standards. Why is this proliferation of solutions an especially serious problem for identity management? As noted by Matt Bishop, “Every access control mechanism is based on an identity of some sort.” Bishop goes on to state that all decisions of access and resource allocation assume that the binding of an identity to the principal is correct **[Bishop 2002]**. Hence, identity is a foundation-level element for

security and accountability decisions, and breakage at this level in design has profound implications for the system's security as a whole. Transactions may employ multiple identity contexts throughout their life cycles, broadening the scope of identity's usage for access.

Piling on to the previously mentioned system-level problems are users' lack of awareness, ability, and tools for managing and propagating their own digital identity information and their lack of ability and technical tools to use in determining the veracity of requests for their personal information. The net result: The emergence of phishing and other attacks targeting these vulnerabilities at the directory, user desktop, and client levels.

To protect identity on the client and server and throughout the system as a whole, software development teams require an overarching understanding of identity's architectural elements and approaches to integrating identity into software systems. Such an understanding will enable them to bridge the chasm that exists between the assumptions made about identities and the actual state that exists in the system with which they are attempting to integrate. The acquisition of knowledge regarding the union of identity elements, behaviors, and constraints and the software user's knowledge and abilities related to the desktop and clients will give software development teams the tools they need to build more robust software based on secure usage of identity.

6.4. System Complexity Drivers and Security

Satisfying business requirements increasingly depends on integrating and extending existing systems. In particular, new development must often be integrated with an existing operational environment. The analysis described in **Section 6.3** concentrated on how an attacker might exploit the Web services interfaces that support a desired integration. Such analysis is representative of current security analysis techniques. Nevertheless, characteristics of the resulting operational environment can not only generate additional security risks, but also constrain the mitigation of those risks.

Some consequences of this tradeoff for security analysis of the distributed operational environments are described in **Table 6–6**. Security risk assessments are affected by unanticipated risks, reduced visibility, and the wider spectrum of failures possible. A number of these factors affect the software development process. The wider spectrum of errors, for example, may require that more attention be devoted to fault tolerance. Other factors may dictate which risk mitigation options can be applied. Factors such as less development freedom, changing goals, and the importance of incremental development all affect how security is incorporated into the software development process.

Table 6-6. *Consequences of Expanded Scope for Security*

Unanticipated risks	The dynamic nature of the operational environment raises software risks that are typically not addressed in current systems. Interoperability across multiple systems may involve resolving conflicting risk profiles and associated risk mitigations among those systems. As work processes cross business units and multiple organizations, change becomes increasingly difficult to control, and any changes might invalidate the existing security analysis.
Reduced visibility	Work processes often involve external, COTS, or legacy systems that cannot be observed or whose behavior cannot be thoroughly analyzed. Testing the subsets of such systems is not sufficient to establish confidence in the fully networked system. That is particularly true when some of the subsystems are uncontrollable or unobservable [Schneider 1999]. In these circumstances, it is much more difficult to distinguish an attacker-induced error from a nonmalicious event. Business requirements may increase the need for interoperability with less than fully trusted systems. Under such circumstances, the security architect cannot have the in-depth knowledge that existing techniques often assume.
Wider spectrum of failures	As noted by Leveson for safety [Leveson 2004], the cause of a system security failure may be not a single event, but rather a combination of events that individually would be considered insignificant. The probability of a single combination of events is typically quite small, but the probability of some adverse combination of events occurring can increase as system size increases. One of the more challenging problems for security analysis involves establishing priorities for identified risks. Increasingly, security risks are associated with high-impact, low-probability events. The relationship frequently applied in risk analysis, Expected Cost = Probability of Event × Impact is not valid for such events.

Less development freedom	Architectural principles provide guidance on how to decompose a system so that the components can later be assembled into a system that meets the security requirements. The decomposition guidance assists in addressing issues associated with component and system interactions. When we integrate existing subsystems or incrementally add functionality to an existing system, most aspects of the decomposition have already been defined. We no longer have the ability to specify a decomposition that best supports the desired system characteristics. At some point, the stress between what we have and what we should have becomes great enough that the only option is reengineering.
Incremental and evolutionary development	Large systems typically emerge from a smaller system by incremental additions of new functionality. A successfully deployed system may encourage new usages that were not anticipated in the original design. While the knowledge that supported the initial design might have been considered sufficient for that context, it may be incomplete for the new functionality and usage.
Conflicting or changing goals	Business usage and underlying technology are typically changing faster than our capability to change the software. Computing systems can be friction points for organizational change.

6.4.1. Wider Spectrum of Failures

Software failure analysis in this context may require a different model of accidents than that used for hardware. Hardware failure analysis typically relies on event-based models of accidents. Such models, with their relatively simple cause-and-effect links, were created in an era of mechanical systems and then adapted for electromechanical systems. The use of software in engineered systems has removed many of the physical constraints that limit complexity and has allowed engineers to incorporate greatly increased complexity and coupling in systems containing large numbers of dynamically interacting components. In the simpler systems of the past, where all the interactions between components could be predicted and handled, component failure was the primary cause of accidents. In today's highly complex systems, this is no longer the case.

While vulnerabilities are often associated with just a single component, the more challenging classes of vulnerability derive from interactions among multiple system components. Such vulnerabilities are difficult to locate and predict, because it may not be possible to model the behavior of multiple systems under all conditions. An unexpected pattern of usage

might overload a shared resource and lead to a denial of service, for example. In addition, multiple factors may contribute to the vulnerability and prove difficult to identify. For example, a vulnerability could arise from a software design that increases the possibility of errors by human operators.

Bellovin observed (in **[Schwartz 2007]**) that the power grid failure in August 2003 was not caused by a single failure but rather by a cascading set of events. Certainly, a race condition that disabled the subsystem that alerted the power grid controllers to potential failure conditions was a significant factor, but that failure by itself likely could have been managed. After that subsystem failure, a combination of operator and management missteps led to a significant time period during which the power grid controllers were not aware of serious transmission line problems. The operators saw no system-generated alerts and had not been told of the alert system failure. During that same period, a second monitoring system managed by an independent organization also suffered a series of failures.

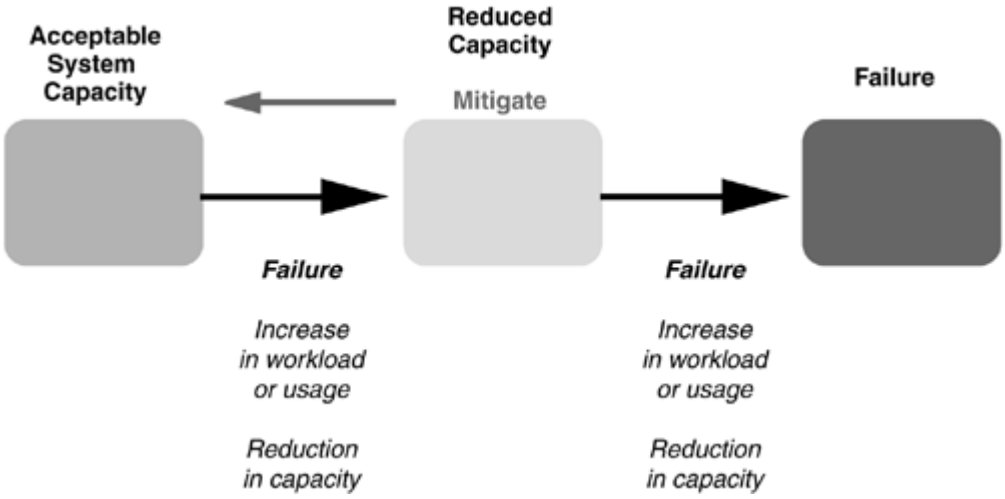
The combination of system, management, and operational errors delayed mitigating the transmission line problems until recovery was impossible. The combination of such events does not have to be concurrent to produce a failure of the system. In this case, the lack of staff mitigation training had existed for some time, which had reduced the capability to mitigate system or operator failures.

Discrepancies between the expected and the actual arise frequently in the normal course of business processes. Discrepancies can be thought of as stresses that may drive a business process into an unacceptable state. Stress types include interactions, resources, and people. Missing, inconsistent, or unexpected data are examples of interaction stresses, whereas resource stresses may include excessive network latency, insufficient capacity, and unavailable services. People stresses can consist of information overload that slows analysis, distraction (too much browsing) and a “Not my job” attitude, which can inhibit effective responses to problems.

Figure 6–2 depicts one way to represent system behavior when the cause of a failure is a combination of events. Think of system capacity as a col-

lective measure of the resources available, both computing and human. A system might be thought of as in a healthy state when sufficient resources are available to manage temporary increases in usage, even with some discrepancies. Over time, however, changes in usage may increase resource loading so that an internal fault that previously could be mitigated now leads to system failure. In the case of the power grid failure, the lack of training reduced the capacity of the system and increased the probability of a mitigation failure.

Figure 6-2. *System failures*



As we integrate multiple systems, we should expect technical discrepancies. Systems developed at different times will inevitably have variances in technology and expected usage. In addition, technical stresses can arise because large distributed systems are constructed incrementally. The functionality of the initial deployment of a system may suggest other applications that were not anticipated in the initial design. Finally, users frequently exploit system functionality in unanticipated ways that may improve the business processes, but their efforts may also stress the operation of components that were not designed for the new usage.

The overall success of a business process depends on how the staff and supporting computing systems handle those discrepancies. Changes in business processes and systems, for example, can introduce new discrepancies. In addition, dealing with discrepancies becomes much more difficult as the number of participants—people and systems—increases. Each participant must cope with multiple sources of discrepancies, and a single discrepancy can affect multiple participants. In this situation, the likeli-

hood that a discrepancy will not be properly managed by some participant increases, with that failure then affecting other participants.

Partitioning Security Analysis

The system complexity associated with business system integration requirements expands the spectrum of development, user, and system management failures that security analysis has to consider. One way to partition that effort is to consider two perspectives for the analysis of work processes that span multiple systems.

The first perspective focuses on the global work process. The compositions of the functions associated with the individual systems certainly must meet the functional requirements for the work process, but the challenge is most likely meeting quality requirements such as those for security, performance, and reliability. How do individual system failures affect the work process? The individual system risks and chosen mitigations may not be compatible with the risk profile desired for the work process, particularly when the individual systems are independently managed or provide services to multiple work processes.

The second perspective is that of a service provider. An essential requirement is to provide the specified functionality, but as with the work process perspective, the quality requirements pose the greater challenge. An individual system has to meet the quality requirements associated with a request, yet there are also risks associated with servicing a request. A requesting system may have been compromised, for example, but system resources must remain protected. Unexpected patterns of usage by a single or multiple requesters may adversely affect the capability to provide a service. Whereas a global work process has limited knowledge about individual systems, a service provider is in an equivalent position with the requesting systems and about the effects that a system action might have on external work process.

Mitigations

• Concentrate first on the first three categories of security failures described in **Section 6.2.1**. With more interfaces to less trusted systems, it is

critical to first deal with known interface vulnerabilities. Pay particular attention to network-based interfaces such as those included in Web services. Web services provide mechanisms to manage security across geographically and even independently operated systems, but the use of those protocols could also be exploited by an attacker. Some of the vulnerabilities associated with Web services are described in **Section 6.3.2**.

The remaining mitigations apply to the general problem of integrating multiple systems.

• One approach to simplifying security across multiple systems is to share essential security services such as user authentication and authorization. Interoperability among multiple systems often results in security problems raised by multiple access control and authentication control points. For example, an operating system supports user authentication and controls access to operating system objects such as files and processes; a database server supports independent user authentication and access control mechanisms for database objects such as tables or individual data items. Identity management refers to an approach to integrating those multiple authentication and authorization mechanisms so that some aspects of identity can be shared among systems. Identity management represents an essential aspect of security functionality. A successful attack on identity management services can enable the attacker to gain enhanced privileges and thereby access confidential information. Security issues for identity management are discussed in **Section 6.3.4**.

• An essential design task for a large system is delegating the responsibilities for meeting security requirements. Delegation of responsibilities goes beyond system components and includes users and system management. For example, a password or private key used for authentication can be compromised by a careless user, so in this sense the authentication responsibilities are shared by users and the system. The user responsibilities might be reduced by using a one-time password mechanism or a biometric device such as a fingerprint 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 occur in advance of the call to the component.

Poor delegation of responsibilities is often reflected by a "Not my job" response and inaction when problems arise. 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, (4) system interactions, (5) tradeoffs among system properties such as performance and reliability, and (6) adaptation over time **[Leveson 2004]**. A risk for security is that it is typically treated as a separate concern, with responsibility being assigned to different parts of the organization that often function independently. That isolation becomes even more problematic as the scope and scale of systems expand. Business integration requirements and the use of technologies such as Web services to support the integration of distributed systems can affect the delegation of responsibilities. It is 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.[3]

• **[BSI 30]** and **[BSI 31]** provide guidance on risk assessment and security concerns for COTS and legacy systems.

A number of organizations are exploring approaches to better manage the risks associated with system complexity. Such practices are on the leading edge of security initiatives and certainly have not been proven. The following suggestions are drawn from those experiences:

• *Operational monitoring.* Failures of some sort are a given for a complex system. A system design usually identifies some set of potentially adverse events and corresponding mitigations, but that set of adverse events is never truly complete. Given this fact of life, it is critical to monitor any unexpected events that fall outside that set. Potential security failures may change because of the appearance of new attack patterns, and changes in usage or system configurations may generate unexpected activity. It can be helpful to analyze system failures that have affected other organizations, because they may identify a similar internal weakness.

The monitoring strategy is similar to that applied to deal with hard-to-solve problems. With this approach, the analysis serves as a learning experience for the staff and increases their capability to respond to future failures and to incorporate that knowledge into system enhancements.

• *Consolidation of failure analysis and mitigations.* The multiplicity of systems and increasing number of possible error states arising from the interactions can overwhelm analysis. The risk in this case is having too many point solutions that mitigate narrowly specified events. Changes in usage could then generate a significant reengineering effort. Failure analysis is done for security, reliability, and safety if applicable. How much of that effort can be consolidated?
Given system complexity and dynamics of usage, it is not realistic to assume that a system is entirely free of vulnerabilities. Instead, error management must be perceived as an ongoing activity for any large system. When an error occurs, there is rarely sufficient information to immediately identify a specific cause, let alone characterize that cause as malicious. The runtime choice of an appropriate mitigation might be based simply on the impact of the failure. If the identified error involves data integrity, the general strategy independent of the cause may be to contain the damage and enable recovery. A recovery from a security breach may be able to take advantage of the general failure mitigations.
An attacker may be able to exploit how a system manages failures—particularly failures in the interfaces with normally trusted components. As an example, one of the causes of the 2003 electric power grid blackout was a race condition in a subsystem that monitored sensor data. While there were no malicious events for that blackout, could an attacker crash such control systems by targeting trusted sensors?

• *Generalization of the problem.* Some 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. Security, reliability, and compliance are all part of business continuity.

Aspects of such guidance could be applied to enterprise architectures that are not geographically distributed. One tactic for supporting business continuity for a system of systems could 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 so that processing can continue asynchronously with an eventual synchronization.

6.4.2. Incremental and Evolutionary Development

Incremental system development affects the design of the software architecture for most of the quality attributes, not just security. A software architecture can realistically support only a limited number of anticipated changes. The consequences of change can be greater for security than the other quality attributes, however, if incremental development introduces vulnerabilities.

The authors of the 2007 National Research Council (NRC) report *Towards a Safe and More Secure Cyberspace* observed the lack of adoption of known techniques for improving system security **[Goodman 2007]**. **Section 4.3.1** describes a number of architectural principles used to enhance software security, such as least privilege, defense in depth, and securing the weakest link. Researchers have found that the use of such architectural principles by system designers and architects correlates highly with the security and reliability of a system **[Goodman 2007]**. Unfortunately, these principles have not been widely adopted. The NRC committee proposed that the primary reasons for that lack of use included the following issues:

• A mismatch between the principles and current development methodologies

• The short-term costs associated with serious adherence to those principles

• Potential conflicts with performance

As stated in the report, an examination of the architectural principles suggests that a serious application of them depends on designers and architects knowing very well and in considerable detail exactly what the software component is supposed to do and under which conditions. All too frequently, however, system requirements and specifications are incomplete. In addition, user requirements may change during development. Likewise, architectural tradeoff analysis among quality attributes such as reliability, security, and performance can lead to revisions of requirements. While incremental development, simulation, and prototyping can enable users to better specify their needs and provide the developer with a better knowledge of how to implement the desired behavior, the use of legacy systems, COTS components, or systems developed and operated by external organizations means that our system knowledge will always be incomplete. Perhaps not surprisingly, the NRC report listed the ability to incorporate security into an incremental development process as an important research topic.

Increasingly, system development depends on integrating existing systems, as reflected in both the Web services and identity management examples. Size complexity can be mitigated by the use of commercial products, the sharing of software-based services among multiple systems, and the reuse of existing software components or systems (i.e., legacy systems). The support for multisystem work processes—particularly those that involve multiple organizations—depends on the ability to create a system of systems. The security issues raised by the design of computing support for a multisystem work process are similar to those associated with the design of a large system in the typical IT environment that uses a significant number of commercially supplied products and legacy systems. In both cases, a new or updated component must merge with an existing operational environment to form a satisfactory operational whole. And in both cases, the systems have not necessarily been designed for the new use or for any additional threats associated with the new use or operating environment.

6.4.3. Conflicting or Changing Goals Complexity

Conflicting goals occur when desired product quality attributes or customer values conflict with one another. There may be conflicts between

portability and performance requirements. In addition, conflicts frequently arise between security and ease-of-use requirements. Meeting the cost requirements for implementing the desired features may increase operational costs and create a conflict between development costs and operational goals. Conflicting goals affect both the developer and the project manager, and there is a learning component for addressing these problems. What are the important interactions among those goals? Which aspects of the software architecture might be affected by a resolution of these conflicts? Do requirements for future needs adversely affect current cost and schedule constraints?

Web services protocols provide mechanisms that describe at runtime which security policy is associated with a transaction. That security policy could describe how authentications are done, how data is encrypted, or which data fields must be signed. This adaptability comes at a cost, however: An application might need to support multiple security protocols, and a number of the security threats described in **Section 6.3.4** could potentially exploit that adaptability.

Changes in goals can generate equivalent problems. The implementation of a new objective may conflict with earlier design tradeoffs. Often changes in goals arise from changes in business requirements, changes in usage of existing functionality, or customers gaining new understanding of their own needs. The tendency is to add but rarely remove functionality for an existing system. Over time, that feature creep and any interaction among features can be source of vulnerabilities.

Changes or conflicts can require some form of experimentation on the part of the user and/or the developer in the form of simulation, scenario analysis, or prototypes. Such problems must be resolved before the final design is attempted. In practice, incremental development methods are often used to address ambiguous or vague functional requirements; such methods can delay decisions about troublesome requirements.

The existence of unanticipated hard-to-solve problems and conflicts and changes in requirements are often just a recognition that our understanding of the problem domain and the tradeoffs among requirements or design options is incomplete when a development project is initiated.

Mitigations

• *Assurance cases.* Changes in requirements or threats and the evolution of usage all require that the security aspects of a system be regularly reviewed. For a large system, that kind of analysis could be daunting, but an assurance case can simplify and improve that task (see **Section 2.4**). An assurance case describes the arguments and evidence that support the security claims for a system. It can be analyzed when changes are made to identify any arguments or evidence that may no longer be valid and hence those claims and associated components that may require a more detailed analysis.

• *Operational monitoring.* Our information is always incomplete for complex systems. The reality of ongoing changes increases the importance of operational monitoring, as described in **Section 6.4.1**.

• *Continuous risk assessments.* Existing risk assessments and security testing procedures should be reviewed and updated if necessary.

• *Flexibility and support for change.* The ability to easily change systems is a general problem. A number of efforts—for example, the Object Management Group's Model-Driven Architecture, Microsoft's Software Factory, and service-oriented architecture (SOA)—have sought to address certain aspects of that problem. Aspect-oriented programming proposes techniques that make it easier to change the behavior of the system as described by the quality attributes. At this point, we are still dependent on good software engineering to anticipate the most likely changes or to find ways to adapt existing components to meet the new requirements. As we noted earlier, at some point the gap between what we have and what we should have will become great enough that the only option is reengineering.

6.5. Deep Technical Problem Complexity

Deep technical problems appear to arise more frequently when the software development team focuses on meeting the quality measures for reli-

ability, performance, and security rather than satisfying a functional requirement. Grady Booch commented on this phenomenon in his Web log on March 22, 2005:

> 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 nonfunctional requirements such as performance and security. **[Booch 2005]**

Because security can be a source of hard-to-solve technical problems for the implementation of a single system, it should come as no surprise that it becomes a rich source of deep problems as we deploy more complex systems and systems of systems. All of the items listed in **Table 6–6** can lead to hard-to-solve security problems. Both usage and potential threats are dynamic factors. Complexity affects both the functional and attacker's security perspectives. The functional perspective captures the obvious requirements for authentication and authorization and hence is likely to be considered early in the development cycle. The attacker's perspective may not receive equivalent attention, yet may be a source of problems that require reengineering for their resolution when identified later in development.

Mitigations

Some general guidelines are applicable for managing hard-to-solve problems.

• The hard-to-solve problems should be tackled first. A deep technical problem can generate extraordinary delays if the solution is postponed, but schedule pressures often lead project teams to work on easy-to-solve problems so they can demonstrate rapid progress. The following quotations from **[BSI 29]**, which refer to the Correctness by Construction (CbyC) method, are a good summary of the justifications for this recommendation:

> When faced with a complex task, the natural tendency is to start with the parts you understand with the hope that the less obvious parts will become clearer with time. CbyC consciously reverses this. As risk and potential bugs hide in the most complex and least understood areas, these areas 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 address the hardest part of a problem at the point with the smallest range of design options. Of course, one could take the fashionable approach and refactor the design; however, designing, building, and incrementally validating a system only to change it because risky areas were not proactively considered is hardly efficient and is not CbyC. Managing risk must be done on a case-by-case basis. Common examples are prototyping of a user interface to ensure its acceptability, performance modeling to ensure the selected high-level design can provide adequate throughput, and early integration of complex external hardware devices.

• Enable the technical staff to concentrate on such problems. When the highly skilled staff that is addressing a deep problem attempts to achieve multitasking, this approach typically delays a solution and affects its quality. The analysis and solution of deep technical problems take time but not many people. A solution for the developer may require both learning and experimentation. That is, a developer needs to identify which techniques might work, where a technique might be an algorithm, an architecture design pattern, or hardware. Under what conditions does the specific technique work? How might it be used for the current problem? Does the technique introduce constraints that adversely affect satisfying other system requirements?

• The existence of hard-to-solve problems can affect the feasibility, risks, and costs associated with specific requirements and hence can influence stakeholder commitments to those requirements. The Incremental Commitment Model explores the effects of hard problems on requirements, project management, and acquisition **[Boehm 2007]**.

• The project manager needs to consider risk mitigations such as alternative implementations at least for the interim, but must recognize that such alternatives may not fully meet quality attribute objectives.

6.6. Summary

The technologies and dynamic nature of the operational environment raise software risks that are typically not addressed in current practice. Security assessments are often done for a point in time, and the techniques are not easily adapted to the more dynamic environment that software now has to address. Vulnerability analysis, for example, 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 have only limited value, however, when the system can be dynamically configured to meet changing operational and business needs.

While security analysis is rarely complete, such completeness is often tacitly assumed. The increasing complexity of deployed systems certainly invalidates such an assumption now. Business requirements increasingly lead to the need to integrate multiple systems to support business processes. The design objective to better support rapidly evolving business requirements or to deal with conflicting or ambiguous functional requirements has led to increased use of incremental development methods. For the project manager, such techniques often translate into hard-to-solve technical problems for meeting the security requirements. General guidance for dealing with these kinds of complex problems should be part of any project plan. System complexity leads to a growing need to explicitly incorporate learning into the project schedule that builds the necessary knowledge base about the operational environment, the dependencies and potential interactions among systems, and the risks that may be associated with proposed designs.

General recommendations include the following:

1. Tackle known interface vulnerabilities first. With systems having more interfaces to less trusted systems, developers should concentrate first on known interface vulnerabilities, such as those found in Web services.

2. Conduct end-to-end analysis of cross-system work processes. With increasing complexity, vulnerability analysis of individual systems is not sufficient. The security analysis of work processes that cross multiple systems has to consider the risks for those processes (including end-to-end analysis) as well as the risks that each work process creates for the systems that support it. In short, security analysis has to account for a wider spectrum of errors.

3. Recognize the unavoidable risk that, with the reduced visibility and control of multiple systems, security analysis will be incomplete. One approach to this situation is to focus first on mitigating the possible impacts of an attack and not on the vulnerabilities that were exploited.

a. Attend to containing and recovering from failures. Assume the existence of discrepancies of some form, whether in systems, operations, or users, during the execution of work processes, particularly as usage evolves. Give increased attention to containment and recovery from failures. These issues should be considered in the context of business continuity analysis.

b. Explore failure analysis and mitigation to deal with complexity. The multiplicity of systems and increasing number of possible error states arising from interactions can overwhelm analysis or generate too many point solutions that mitigate narrowly specified events. Explore how security could take advantage of a consolidated failure analysis and mitigation effort.

4. Coordinate security efforts across organizational groups. Security is typically treated as a separate concern, with responsibility often being assigned to independent parts of the organization (such as development and operations). It is 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 fallen through the cracks. This separation becomes even more problematic as

the scope and scale of systems expand. Vulnerability analysis and mitigations should be integrated across organization units, users, technology, systems, and operations.