Supply-Chain Risk Management: Incorporating Security into Software Development

Carol Woody

Robert J. Ellison

March 2010

ABSTRACT: As outsourcing and expanded use of commercial off-the-shelf (COTS) products increase, supply-chain risk becomes a growing concern for software acquisitions. Supply-chain risks for hardware procurement include manufacturing and delivery disruptions,"Supply-Chain Risk Management (SCRM) is a discipline of Risk Management which attempts to identify potential disruptions to continued manufacturing production and thereby commercial financial exposure." [Wikipedia 2010] and the substitution of counterfeit or substandard components. Software supply-chain risks include third-party tampering with a product during development or delivery and, more likely, a compromise of the software assurance through the introduction of software defects. This paper describes practices that address such defects and mechanisms for introducing these practices into the acquisition life cycle. The practices improve the likelihood of predictable behavior by systematically analyzing data flows to identify assumptions and using knowledge of attack patterns and vulnerabilities to analyze behavior under conditions that an attacker might create.

This article was presented as a paper at the Hawaii International Conference on Systems Sciences (HICSS-43).

INTRODUCTION

Software is rarely defect-free, and many common defects such as "improper input validation"[1] as defined by the Common Weakness Enumeration (CWE), a list of software weakness types, [Mitre 2010] can be readily exploited by unauthorized parties to alter the security properties and functionality of the software for malicious intent. Such defects can be accidentally or intentionally inserted

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

[1] "Improper input validation," CWE-20, is the highest ranked software error on the 2009 CWE Top 25 list of the most dangerous programming errors.

into the software, and subsequent acquirers and users have limited ways of finding and correcting these defects to avoid exploitation.

Participation in the software supply chain is global, and knowledge of who has touched each specific product or service may not be visible to others in the chain. At each step in the software supply chain, an organization acquires software products or services from suppliers. Such software products or services may support software development or may be incorporated into a product or service that is supplied to another organization.

Supply-chain risks can be introduced at any point in the supply chain and may be inherited by each subsequent acquirer. Therefore consideration of supply-chain risks should begin as early as possible in the acquisition life cycle. An acquisition such as shown in Figure 1 has an extensive inheritance tree but only has direct control over the shaded oval, and hence inheritance is a significant risk.

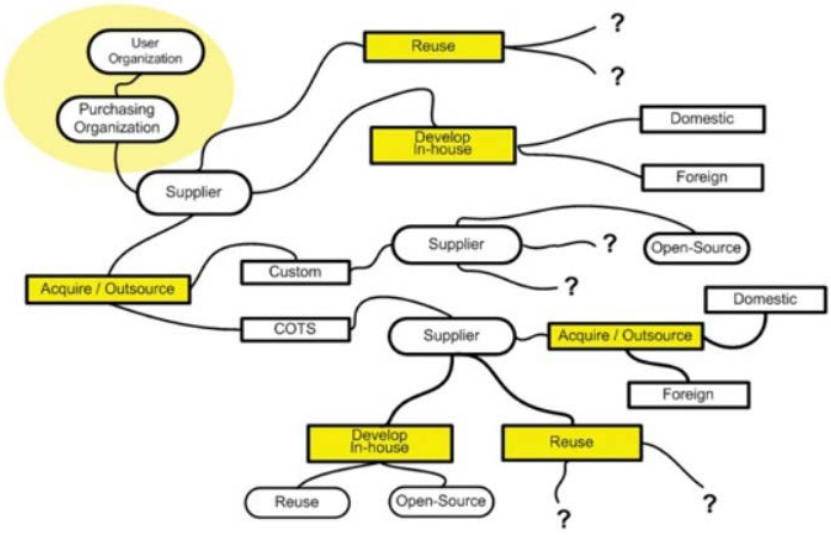

Figure 1: Potential Software Supply-Chain Paths (Initiation and Development) [DHS 2010]

Each of these organizational layers can be responsible for inserting defects for future exploitation. This view of the supply chain does not include the full acquisition life cycle.

There is a subsequent supply-chain segment that involves the operational deployment, use, and eventual disposal of the delivered product, as outlined in Figure 2. Suppliers play a role in this segment that can lead to software supply-chain security risk through the delivery of sustainment upgrades and configuration changes. Coding and design defects identified and reported as vulnerabilities may require patches and special security monitoring to prevent compromise.

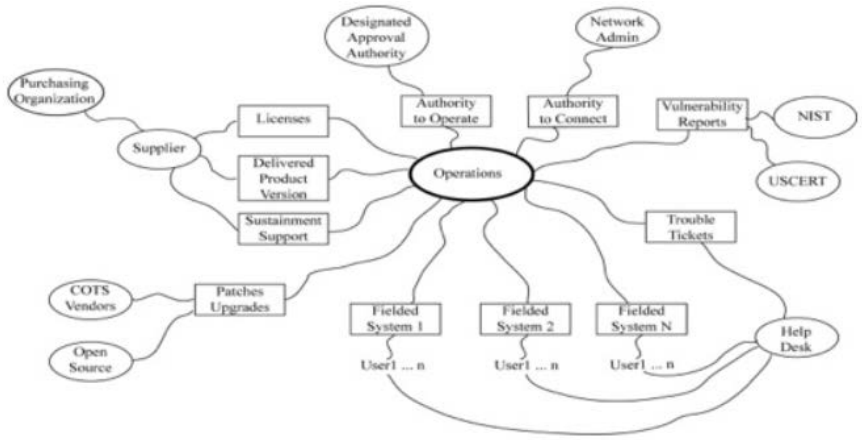

Figure 2: Supply Chain Including Operations

The following definitions are needed for consistency of usage:

- Supplier: an organization involved in providing a product or system to another organization—includes distributors, transporters, and storage facilities as well as the organization directly responsible for creating product or system content
- Supply chain: all suppliers contributing to the content of a product or system or having the opportunity to modify the content

SOFTWARE SUPPLY-CHAIN RISK

Software acquisition has grown from the delivery of standalone systems to the provisioning of technical capabilities integrated within a larger system-of-systems (SoS) context. This integration extends supply-chain risk. Software security defects in any of the products or services presents a potential supply-chain security risk to all participants of the SoS.

Of critical concern in this highly interconnected software environment is the risk that an unauthorized party would use a defect to change a product or system, adversely affecting its security properties and expected behavior. These software security risks are introduced into the supply chain in many ways, such as

- coding and design defects incorporated during development, which allow unauthorized access and execution of protected system functionality and the introduction of code by unauthorized parties after the product or system is fielded

- improper control of access to a product or service when it is transferred between organizations (failures in logistics and configuration control)
- insecure deployment of a product or service when it is fielded (defective configuration and distribution)
- operational changes and use of the fielded product or system that introduce coding defects or configuration changes that allow security compromises (configuration control and patch management)

Software supply-chain risks exist at any point where organizations have direct or indirect access to the final product or system through their contributions as a supplier. Without mitigation, these risks are inherited from each layer in the supply chain including the final product or service, increasing its likelihood for a security compromise.

Reduction of supply-chain security risk requires controlling ways in which security risks can be introduced into the product or service through attention to all of the following within the acquisition life cycle

- supplier capability—supplier's ability to make sure they have good security development and management practices in place
- product security—an assessment of the risk of the product for critical security compromise and mitigation requirements
- product logistics—control of access to the product in transit at each step in the supply chain
- operational product control—the operational use of the product or service to ensure appropriate configuration and monitoring controls that are necessary to reduce the risk that unauthorized changes remain in place as the product and use evolve over time

Supply-chain risk mitigations can consist of extending a developer's product logistics practices to subcontractors. In other instances, practices such as software design reviews or security testing that are adequate for internal development have to be revised to address supply-chain risks.

Embedding software assurance into supply-chain management must include identifying the risks that require action and applying the necessary mitigations at the appropriate phase of an acquisition life cycle, such as shown in the first column of Table 1. All steps in the acquisition life cycle (initiation, development, deployment, operations/maintenance, and disposal) are affected. The second column of Table 1 identifies management activities that should be done to focus proper attention on supply-chain security risk for each step. Supply-chain risk

mitigations across the full acquisition life cycle must become a shared responsibility of the purchasing organization, suppliers, and operations management.

Each acquisition must perform an initial assessment to identify the relevant software supply-chain risks that can be addressed. For example, in the purchase and use of a COTS or open source product, the acquirer will be unable to address specific development risks due to limitations of information about product security. When enhancing existing products, options for suppliers can be hampered by legacy decisions, and development must fit within the existing operational constraints. For acquisitions that are already underway, mitigation options must fit within constraints of contract cost and schedule limitations.

Table 1: Sample Acquisition Life Cycle

Step	Sample Supply Chain Management Activities
Initiation	Initial risk assessment Develop Request for Proposal Plans for monitoring suppliers Select suppliers
Development	Manage Acquisition Maintain awareness of supplier
Operations and Maintenance	Incident handling Review operational readiness Monitor component/supplier
Disposal	Evaluate disposal risks Mitigate risks during disposal

SUPPLY-CHAIN RISK ANALYSIS

A software supply-chain risk can be a defect in the delivered software or in the default installation or configuration that an attacker can exploit. Such a defect could be caused by a failure in the integrity of the supply chain, whereby an unauthorized party maliciously changed the software package. This type of defect can also be the result of a design or coding error during normal development.

A discussion of system security often includes firewalls, authentication issues such as strong passwords, or authorization mechanisms such as roll-based access control, but the defects that typically enable an attack are not in the code that directly implements security.

Most attacks occur in the code that implements the functionality. The CWE[2] classifies security bugs and documents the design and coding weaknesses associated with attacks. The CWE list is dominated by errors in the functional software. As an industry-wide effort reduced operating system and network security vulnerabilities, applications became the next attack target. Application security has often been ignored in part because of the faulty assumption that firewalls and other perimeter defenses could protect the functional code. The problem is further compounded as application developers without specific security training are typically unaware of the ways that their software that meets functional requirements could be compromised. Security software is usually subject to an independent security assessment that considers the development history as well as the design and operation. There is no equivalent effort applied to the security of the functional code.

Security for functional development is getting increased commercial attention. Two events are significant. First, Microsoft started their Trusted Computing Initiative in 2002 and second, then published The Security Development Lifecycle (SDL) in 2006 [Howard 2006]. Measures of effectiveness for a particular software development practice are difficult to implement since the use of a control group is impractical because of costs and limited resources. Comparisons are typically available based on historical data. In 2003 Microsoft compared Office 2003 SQL Server, 2000 Service Pack 3, and Exchange Server 2000 Service Pack 3 with earlier efforts. The products whose development had been influenced by the early efforts of the Trusted Computing Initiative showed reduced vulnerability rates even with ad hoc development processes. The SDL documented the development processes for broader consistency of use.

Today, over 25 large-scale software security initiatives are underway in organizations as diverse as multi-national banks, independent software vendors, the U.S. Air Force, and embedded systems manufacturers. The Software Assurance Forum for Excellence in Code (SAFECode), an industry-leading non-profit organization that focuses on the advancement of effective software assurance methods, published a report on secure software development [Simpson 2008]. SAFECode members include EMC, Microsoft, Nokia, Adobe, SAP AG, and Symantec. In 2009, the first version of The Building Security In Maturity Model (BSIMM) was published [McGraw 2009]. BSIMM was created from a survey of nine organizations with active software security initiatives that the authors con-

---

[2] Sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security

sidered to be the most advanced. The nine organizations were drawn from three verticals: financial services (4), independent software vendors (3), and technology firms (2). Those companies among the nine who agreed to be identified include Adobe, The Depository Trust & Clearing Corporation (DTCC), EMC, Google, Microsoft, QUALCOMM, and Wells Fargo.

Increased attention on secure functional components has influenced security testing practices. All of the BSIMM organizations do penetration testing, but there is increased use of fuzz testing. Fuzz testing creates malformed data and observes application behavior when such data is consumed. An unexpected application failure is a reliability bug and possibly a security bug. Fuzz testing has been effectively used by attackers to find vulnerabilities. Fuzz testing applied during development has the same objective. Microsoft's SDL includes requirements for fuzz testing. One requirement is that an application that reads a file is tested with 100,000 automatically generated malformed entries. If the application fails unexpectedly, it is retested after repairs against a different stream of malformed files. The files that generated failures are then archived and used to test new versions. Fuzz testing does not generate purely random data but typically uses knowledge about the protocol, file format, and content values. For example, in 2009, a fuzz testing tool generated XML formatted data that revealed an exploitable defect in widely used XML libraries [Codenomicon 2009].

Microsoft's security testing provides additional evidence for the benefits of the SDL. At Microsoft about 20 to 25 percent of security bugs are found via fuzz testing [Howard 2006]. The percentage of such security bugs found in code developed under SDL has been reduced. The vast majority of such bugs are now in old code not developed under the SDL.

Acquisition and Operational Contexts

The analysis of supply-chain risks and mitigations goes beyond product and supplier assessments and has to include deployment and usage. A typical software product provides more functionality than is actually required. An attacker may be able to exploit those unused features; a deployment needs to control usage. The required usage also affects risks and mitigations. For example, a required product feature that requires the processing of JavaScript or XML-formatted input expands the scope of the analysis of supply-chain risks and mitigations. Product functionality is typically the primary driver for selection, and a fully-functional product may have residual supply-chain risks that have to be mitigated during deployment.

The two techniques proposed in this section that should be considered in an acquisition, attack surface and threat modeling, are essential parts of Microsoft's

SDL. The objective of proposing these is not to promote them specifically, but to use them as examples of the general approach and objectives of the effective practices in the BSIMM.

Concentrate the Analysis—An Attack Surface

The scope of supply-chain risk analysis can be overwhelming. Defective software could appear anywhere along the supply chain. Malicious code can be inserted anywhere in a system. How can progress be measured?

An approach to managing the scope of the analysis arose from pragmatic considerations. Howard in 2003 observed that attacks on Windows systems typically exploited a short list of features such as open ports, services running by default, services running as SYSTEM, dynamically generated web pages, enabled accounts, enabled accounts in admin group, enabled guest accounts, and weak access controls [Howard 2003a]. Instead of counting bugs in the code or the number of vulnerability reports, Howard proposed to measure the attack opportunities, a weighted sum of the exploitable features. An attack-surface metric is used to compare multiple versions or configurations of a single system. It cannot be used to compare different systems.

Howard's intuitive description of an attack surface led to a more formal definition with the following dimensions: [Howard 2003b]

- targets—data resources or processes desired by attacker (A process could be a web browser, web server, firewall, mail client, database server, etc.)
- enablers—the other processes and data resources used by attacker (e.g. web services, mail client, or having JavaScript or ActiveX enabled. Mechanisms such as JavaScript or ActiveX give the attacker a way to execute their own code.)
- channels and protocols (inputs and outputs)—used by attacker to obtain control over targets
- access rights—control is subject to constraints imposed by access rights

A simple attack-surface example is show in Figure 1 for an application that accepts user input and inserts that input into a database query.

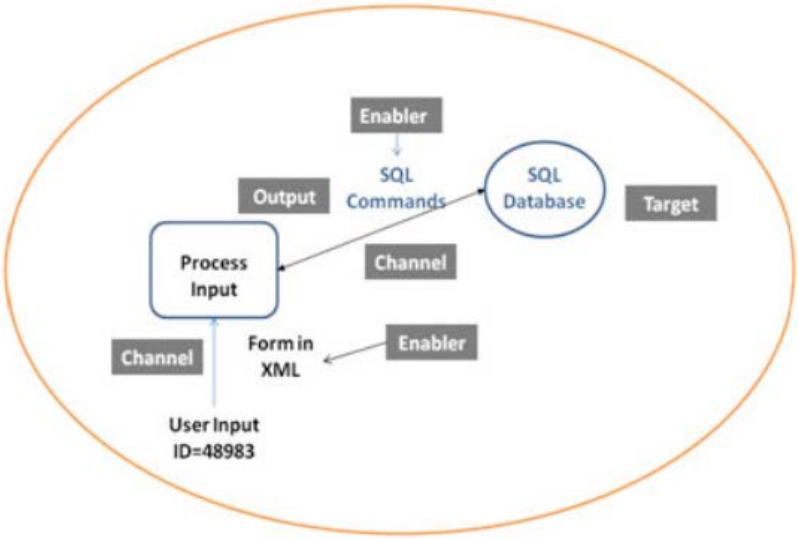

Figure 3: Attack-Surface Example

There is not an accepted way to calculate a numeric value for the area of an attack surface from these factors. The attack-surface area in Howard's calculation is the sum of independent contributions from a set of channel types, a set of process-target types, a set of data target types, and a set of process enablers, where each type is given a weight and all are subject to the constraints of the access rights.

The concept of an attack surface is useful even if an explicit numeric value is not calculated. An increasing number of applications are XML-enabled, and XML is an attack-surface enabler. Such applications have a supply-chain risk. For example, the Finnish firm Codenomicon reported in August 2009 that vulnerabilities existed in XML libraries from Sun, Apache Software Foundation, Python Software Foundation, and the GNOME Project (instances likely in the millions) that could result in successful denial-of-service attacks on applications built with them [Codenomicon 2009]. The particular vulnerability was first identified and patched in Microsoft systems in 2002.

Let's consider the example in Figure 3. The attack target is a database with the attack-surface weight determined by the type of data. A database with credit card or other customer financial information would have a high weight. The channels and protocols are significant contributors to the attack surface for this example. A function that accepts a user-filled form with entries that are used to query a database should be given a high weight since such types of channels have a history of exploits that gave attackers access to confidential information.

An attack surface supports consideration of software assurance risk in several ways.

- A system with more targets, more enablers, more channels, or more generous access rights provides more opportunities to the attacker.
- Feature usage of the product influences the attack surface for that acquirer. The attack surface can be used to compare the opportunities for attacks when usage changes.
- An attack surface helps to focus the analysis on the code that has to be trusted. A reduced attack surface also reduces the code that has to be evaluated for threats and vulnerabilities.
- For each element of a documented attack surface, the known weaknesses and attack patterns can be used to mitigate the risks.
- The attack surface supports deployment as it helps to identify the attack opportunities that could require additional mitigation beyond that provided by the product.

An attack surface is of value to a vendor. Microsoft calculated the relative attack surfaces for Windows Server 2003 as 174, while the value for Windows 2000 was 350. A lower value should translate into lower maintenance costs and improved user perception of security [Simpson 2008].

Threat Modeling

The creation of an attack surface helps to focus analysis but does not identify the security risks and possible mitigations for the functional components. Threat modeling, which is part of Microsoft's SDL, is a systematic approach to identifying security risks in the software and rank them based on level of concern [Howard 2006, Swiderski 2004].

Threat modeling constructs a high-level application model (e.g. data-flow diagrams), a list of assets that require protection, and a list of risks. For this discussion, we assume that it also produces a list of mitigations that is not necessarily a requirement in Microsoft's characterization of the practice.

A detailed walk-through of a data flow considers the deployed configuration and expected usage, identifies external dependencies such as required services, analyzes the interfaces to other components (inputs and outputs), and documents security assumptions and trust boundaries, such as the security control points [NIST 2009].

As noted in [Steingrubel 2009] undocumented assumptions have caused significant system failures. The problem often occurs when we compose systems with

incompatible assumptions. For example, a classic security problem with legacy systems is that they were often designed under the assumption of operating in a trusted and isolated environment. That assumption is violated if such legacy systems are used as a back end for a Web-based public-facing front end. Threat modeling a multi-system data flow can identify such assumption mismatches.

Consider the attack-surface example shown in Figure 3. Threat modeling analyzes the data flow associated with that figure. If the user submits an ID value of 48983, then the output from the input routine is likely a database command, such as

SELECT ID name salary FROM EMPLOYEES

WHERE ID=48983

in response to which the server returns

48983 Sally Middleton $74,210.

For properly formatted input, the data flow will successfully execute and return the desired data or an error if the ID is not found.

Threat modeling analyzes how this data flow could be compromised by malformed input. In this example, user input, the ID, has been used to create the database query. Threat modeling draws on known attack patterns and vulnerabilities. A data flow where user input is part of a command that will be executed by another process automatically raises a “red flag” for a security-knowledgeable developer given the extensive history of software defects associated with such data flow. Table 2 shows an example where user input is used to create a full name that will be used to access a file in folder A. If that input can include file system commands such as “../” then the user may be able to access files outside of the intended folder.

Table 2: File System Command

Input	Command	Comment
Costs	C:\ACosts	Access file Costs
..BCosts	C:\A..BCosts	Changed folder

Inputs for the examples in Table 3 are URLs that an attacker has convinced a user to submit. Web server vulnerabilities associated with the processing of those URLs can adversely affect the user and the web server. The attacker’s objective with the first example is to have the bogus site displayed and hopefully inherit

the user's trust. Such a URL should be rejected by the trusted site's web server software, but there are numerous examples where server software accepts such URLs and loads the bogus site. Web server execution of embedded JavaScript in the second example can pass user data on that server to the attacker.

Table 3: Web Server URLs

Command	Comment
http://trustedsite/...../bogus_site	Redirection may lead user to trust bogus site.
http://trustedsite..../JavaScript	JavaScript embedded in the URL may give attacker access to user data on trusted site.

In the database example, malformed data could include database commands. The symbol | is interpreted by a SQL database server as a logical OR. Input of 48983 | 1= 1 would download information for all employees, as the selection criteria ID = 48983 or 1 = 1 is always true. The risk of this defect is high, as variants of it have been used in attacks that illegally download credit card data.

A number of recent efforts document and classify known attacks and vulnerabilities.

- Common Vulnerabilities and Exposures (CVE)
- Common Weakness Enumeration (CWE)
- Common Attack Pattern Enumeration and Classification (CAPEC)

There are usually yearly lists of the top 25 vulnerabilities associated with the CWE. A discussion [Howard 2009] of a 2009 list includes these CWE categories:

- Improper Input Validation (a buffer overflow)
- Failure to Preserve SQL Query Structure (SQL injections such as the database example)
- Failure to Preserve OS Command Structure ("../" example)
- Execution with Unnecessary Privileges
- Code Download without Integrity Check

Threat modeling is never complete and cannot guarantee that functional code is free of security-related defects. It is based on current knowledge. New attack techniques may be applicable to code that was once thought to be secure.

The improved software assurance that results from defect identification and mitigation associated with threat modeling or equivalent risk analysis techniques reduces the overall supply-chain risk for those using the software component.

The benefits of threat modeling for the developer are documented in [Howard 2006]. Those benefits include requiring development staff to review the functional architecture and design from a security perspective, contributing to the reduction of the attack surface, and providing guidance for code reviews and security testing. The threat model identifies the components that require an in-depth review and the kinds of malformed input that might expose defects. For the database example, security and penetration testing should include SQL injection and may be able to use commercial tools designed to find SQL-injection vulnerabilities.

A potential range of risks can be assembled using the Common Attack Pattern Enumeration (CAPEC). This source describes many of the ways in which software has been successfully attacked and references the coding weaknesses that allowed these attacks. STRIDE is a threat model developed by Microsoft for their use in evaluating the products they develop. Their security analysts have determined the six categories of threats that require critical consideration:

1. spoofing identity—an attacker gaining access by imitating a legitimate user
2. tampering with data—unauthorized modification of data as it flows among segments of code and computers
3. repudiation—threats associated with a user allowed to perform an action without sufficient evidence that the action occurred
4. information disclosure—inappropriate exposure of information to parties that should not see it
5. denial of service—attacks that could make a system temporarily unavailable or unusable
6. elevation of privilege—ways in which a user could inappropriately increase their access to system capabilities

For more complex acquisitions, threat models such as STRIDE may not be sufficient. Use of the Survivability Analysis Framework (SAF) (see Figure 4) built to evaluate the quality of the linkage among roles, dependencies, constraints, and risks for critical technology capabilities, can provide an effective means of identifying and addressing threats in these highly complex structures [Ellison 2009]. It is not uncommon for the supplier to provide systems and software composed of distributed, interconnected, and interdependent networks that cannot be effectively evaluated individually. The SAF constructs an operational model for a work process that provides a context in which to reason about a wide spectrum of failures: technology, software, systems interoperability, operational, and human.

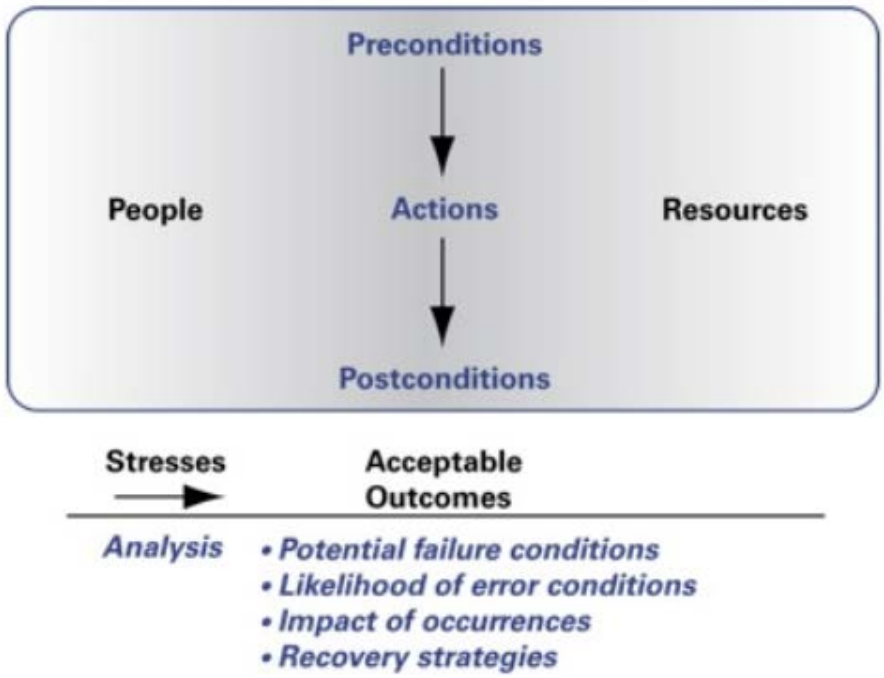

Figure 4: Survivability Analysis Framework

SAF characterizes the specific actions of each step in a business process and the linkages between each step. By evaluating successful business process completion and considering ways in which success could be jeopardized, SAF provides a structured approach to capture the stresses that may impact a business process. It also provides opportunities to analyze whether the stress-handling approaches adopted by a step are compatible with subsequent business process steps.

Product Delivery

At a minimum, the product should be delivered to the acquirer in a controlled manner to ensure that what is received has not been tampered with in any way by the delivery mechanism. In addition, delivery should include a description of the attack surface that was minimized by the supplier and the threat model used by the supplier to achieve a secure result.

The acquiring organization should arrange for verification and validation of the delivered product and accompanying data to ensure the product is as described before accepting the product.

The acquirer should review threat modeling activity and if possible verify the mitigations in the source code. Additional testing should focus on inducing failures for elements of the attack surface such as doing penetration testing and ad-

vanced fuzz testing—a software testing technique that provides random data to the inputs of a program outside of the normal ranges of expected input (described earlier in this article).

When the supplier does not provide adequate documentation on an attack surface and threat mitigation, the acquirer can develop this information from the supplier's product document if it is sufficiently complete. Without a documented threat analysis and mitigation report, the testing will have to be expanded with more emphasis on well designed fuzz testing to identify potential product risks that will have to be accommodated in usage.

Threat modeling should be done as part of the integration of the acquired software with the existing operational system. For the acquiring organization, threat modeling should lead to a delivered product with a reduced number of vulnerabilities, but there may be vulnerabilities created by the integration of the new software with the acquirer's deployed systems. The acquirer should also apply threat modeling to integrated systems or a system-of-systems to identify mismatches among design, development, or operational assumptions associated with the acquired software that could be exploited.

Product Use

The acquirer can deploy the product within a production environment or integrate it into a larger product for delivery to another acquirer. In either case the acquirer assumes responsibility for preserving product trustworthiness within the context of its usage. The following changes could occur and require attention to the usage of the product so as to not jeopardize the level of security provided by the supplier:

- usage changes that affect the attack surface and threats
- threat changes that invalidate supplier security mitigations, vendor upgrades/patches, and local configuration changes

The attack surface and the mitigations introduced by threat modeling are not static artifacts. Attackers can introduce new techniques that are applicable to software that was thought to be secure. The attack surface and associated threat modeling should be periodically reviewed. New technologies should be reviewed more frequently given the relatively short history of exploits that are available to guide threat modeling.

APPLYING THE APPROACH

Our team performed a preliminary review of the supply-chain risks for a selected Department of Defense (DoD) program using the framework described in the previous sections of this article. The program selected for review is in the development phase of the acquisition life cycle. Our review was limited since program funding was not available for the contractors to meet with us and most of the documents were not considered releasable to external parties. We were able to review the program's Software Development Plan (SDP) and conduct a group interview with key government members of the program office. All quotes in this section are taken from the SDP.[3]

As a federally funded research and development center (FFRDC), the SEI has reviewed many DoD programs for a wide range of reasons. Our experience with the program we reviewed indicates that it is typical in its approach to software assurance and supply-chain risk. Many organizations have focused on system security protections to prevent intrusion and have not developed approaches that mitigate security defects in functional software. As systems become highly interoperable, the protection level provided by system security mechanisms diminishes. Without effective management, the software supply chain further increases the opportunity for the introduction and attack of software security defects.

The supply-chain structure for the program we reviewed consists of a prime contractor who is focused on system and network management. There is a major subcontractor to the prime contractor handling the functional software and information assurance software. Three additional development companies are addressing specialized software needs and are subcontractors to the major software subcontractor. "The… Program deployment software consists of commercial off-the-shelf (COTS) software, Government off-the-shelf (GOTS) software, Non-developmental software from Independent Development (ID)/Independent Research and Development (IRAD) projects, and Free and Open source Software."

The COTS products were selected by the vendors as part of their initial response to the RFP. The contract is cost-plus and based on functionality to address the mandated requirements. The architecture (SOA) was also part of the contractor bid. A great deal of the acquisition is outside of visibility of the DoD and the prime contractor. Exposure to the functionality is provided through periodic formal reviews specified in the Software Development Plan.

---

[3] This document is the property of the program our team reviewed, and access is restricted.

Evaluation of the Program Supply-Chain Risks

The approach to addressing software assurance for the supply chain can be thought of in four parts: supplier capability, product security, product logistics, and operational product control. In the remainder of this section, we discuss each of these parts.

Supplier capability. Because the project is already in development and the contractor selection was complete, we could not view information related to how that selection was made to identify considerations for supply-chain risk. However, we asked about supplier capabilities for the production of software with minimal security defects. The answer was that the supplier was only handling classified information and had no bearing on their ability to produce secure code.

The supplier's staff are required to complete background checks and clearances to work in a top secret environment. Developers are exposed to security awareness material, and an annual security briefing is conducted by security staff to include classification/declassification guidelines and marking requirements.

The supplier's security staff identified security solutions for system access and authentication but those solutions do not extend into the software. The standard DoD solutions of Public Key Infrastructure (PKI), certificates, role-based access controls, and intrusion detection systems are included in the software architecture requirements. These can be successfully or inappropriately supported by software when they are implemented, depending on the skill of the developers. Security requirements mandate the use of application code signing, which at least provides a level of accountability if defects are appropriately tracked back to the source.

Based on information from the supplier's SDP, the skill focus is on a software developer's ability to create new software. The following skills, which cover a typical list of programming capabilities, are required at various levels of experience, but it is not clear that knowledge of secure use of these tools beyond password control is expected: XML, C, C++, Java, CORBA, UNIX, ClearCase, Windows, Linux and Solaris, network administration, TCP/IP, X/Motif, DII COE, Simple Network Management Protocol (SNMP), Agent Technology, 3D(LDAP)v3 interfaces, OOA/OOD, UML, and COTS integration. Without proper training, programmers using these tools can accidentally create code that allows all of the common software attacks identified by CWE.

Program coding standards specified in the supplier's SDP require use of Java, C, and C++ but do not include any consideration of secure coding standards for these languages (e.g. C and C++ Secure Coding Standards).

In addition, the subcontractors are building code using code generation tools (e.g. Spring Framework) that will create code that allows many of the security weaknesses unless programmers have been trained to avoid these problems. The use of technology to generate code to improve the development process can increase the supply-chain risk if security weaknesses are not properly addressed.

Product security. There were no indications that the supplier considered security beyond specific system requirements (PKI, certificates, role-based access controls, and intrusion detection systems). Industry regulations lack security practices for software, which does not ensure security is considered in the requirements.

Interviews our team conducted with the supplier also indicate a great deal of quality-of-service monitoring (for a SOA environment) is built into the infrastructure that can support security. There is extensive monitoring to ensure software is well behaved. Most of the messaging will be in XML, which can be structurally validated.

The supplier's development team will apply software engineering principles such as

- isolation of interfaces to minimize data visibility and maximize information hiding
- isolation of performance sensitive software
- encapsulation of COTS
- encapsulation of hardware and change points

However, COTS tools used in development to generate code and COTS execution software that makes up the SOA infrastructure cannot be readily isolated. Many of these tools have security vulnerabilities. There are no formal acceptance criteria in place for COTS. Control of patches and the COTS refresh process is the responsibility of the contractor until the system is fielded, but this investigation indicated that no level of security currency is required. The impact of a vulnerability on an isolated service will be quite different from one that is heavily reused. Of greater concern will be for vulnerabilities that are within the SOA infrastructure on which all services rely. Our team did not see that there was planning for control of these varying levels of criticality.

"Testing shall be designed to not only show that all requirements are reflected in the software's behavior, but also try to demonstrate that the software's behavior is completely reflected in the design and requirements (no undocumented behavior)." The use of the term "try" does not provide a strong indicator of the level of

success expected. This, coupled with limitations in encapsulation, indicate the potential for a very broad attack surface.

Because the prime contractor and subcontractors compete directly on other government contracts, exposure of identified problems through the supply chain has been problematic. Each supplier will inform the government but will not necessarily share that information with potential competitors. Though a formal reporting mechanism for problems has been formulated in the SPD, it is unclear how well this is being utilized based on the relationships of the contract participants.

The reviews are conducted by a quality control team, an independent auditing capability with a "separate reporting structure outside of the … program and engineering management structure." Theoretically this should provide the best level of information. However, our interviews indicate that QC does not have the knowledge to cover everything. While additional outside support is being sought, cost is a limiting factor.

Product logistics. An informal control process is in place for all software elements during development, and all vendors share a central repository and other development tools. A formal control process under the control of a build coordinator is defined for the baselines of all products that move into the system testing phase.

For limited user testing (LUT) the prime contractor hired a firm to formally control the software products from their environment to the specified user platforms and back.

The contractor environments are audited for a top secret security rating. SafeCode has proposed a Software Supply Chain Integrity Network [Simpson 2009] that addresses many of these issues.

Operational product control. The control of security patches and COTS refresh passes to post deployment support beyond implementation. Patches are applied as soon as they are tested. Maintaining an effective SOA environment with minimum security risk that is highly dependent on COTS products customized for the SOA environment will present supply-chain risks beyond the normal vulnerability management concerns.

The supplier carefully controls license management, and control is transferred to the government at operational implementation. This provides the government with access to support from the vendor, which includes security patches.

Up to seven kinds of logging are available, but performance considerations must be factored into the use of each kind. The supplier must still establish decisions about what is sufficient. In addition, use of the logs will require some level of responsive monitoring that will be difficult, if not impossible, to do manually.

The fielded system is budgeted for a five-year replacement of infrastructure components instead of the usual 20 years, which provides some protection against security deterioration. Old versions of hardware and software even if patched frequently carry defects that are never fixed.

Evaluation of the Review

The analysis approach our team used provides a structure for the identification of what areas of supply-chain risk are relevant to a project. Acquisition concerns will vary depending on where in the acquisition life cycle the evaluation is done. Consideration should be given to a broader range of supply-chain related risk than a general risk assessment or security control review would provide.

By focusing on the specific areas of supplier capability, product security, product logistics, and operational product control, a broad range of program practices can be considered in relationship to supply-chain risk.

CONCLUSION

This article considered practices that can reduce the likelihood of vulnerabilities in acquired software. The calculation of an attack surface and the application of threat modeling are examples of practices that evaluate software quality from an attacker's perspective. That perspective continues for security testing where attackers use techniques such as fuzz testing use malformed input to look for vulnerabilities. The analysis of an attack surface may lead to a revision or removal of a desired but high risk feature. Risk assessments such as threat modeling should be part of architecture development as well as for detailed design and coding. Our team applied the supply-chain secure coding analysis to one acquisition. A second trial is planned for the spring of 2010.

REFERENCES

[Codenomicon 2009] Codenomicon. Codenomicon DEFENSICS for XML Finds Multiple Critical Security Issues in XML Libraries. (2009).

[DHS 2010] Department of Homeland Security. Acquisition and Outsourcing Working Group. (2010).

[Ellison 2009] Ellison, Robert, Goodenough, John, Weinstock, Charles, & Woody, Carol. Survivability Assurance for System of Systems. (CMU/SEI-2008-TR-008.) Software Engineering Institute, Carnegie Mellon University, 2009.

[Howard 2003a] Howard, Michael. "Fending Off Future Attacks by Reducing Attack Surface." Proceedings of the Workshop on Advanced Developments in Software and Systems Security, Taipei, December 2003.

[Howard 2003b] Howard, Michael, Pincus, Jon, & Wing, Jeannette. Measuring Relative Attack Surfaces. (2003).

[Howard 2006] Howard, Michael & Lipner, Steve. The Security Development Lifecycle. Microsoft Press, 2006.

[Howard 2009] Howard, Michael. "Improving Software Security by Eliminating the CWE Top 25 Vulnerabilities." IEEE Computer. (June/July 2009): 68-71.

[McGraw 2009] McGraw, Gary, Chess, Brian, & Migues, Sammy. The Building Security in Maturity Model (2009).

[Mitre 2010] The Mitre Corporation. 2010 CWE/SANS Top 25 Most Dangerous Programming Errors. 2010.

[NIST 2009] National Institute of Standards and Technology, "Recommended Security Controls for Federal Information Systems and Organizations," NIST Special Publication 800-53 Revision 3, 2009.

[Simpson 2008] Simpson, Stacy, ed. Fundamental Practices for Secure Software Development: A Guide to the Most Effective Secure Development Practices in Use Today. (2008).

[Simpson 2009] Simpson, Stacy, editor. The Software Supply Chain Integrity Framework. (2009).

[Steingrubel 2009] Steingruebl, Andy & Peterson, Gunnar. "Software Assumptions Lead to Preventable Errors." IEEE Computer. August/September (2009): 84-87

[Swiderski 2004] Swiderski, Frank & Snyder, Window. Threat Modeling. Microsoft Press, 2004.

[Wikipedia 2010] Wikipedia.com. Supply Chain Risk Management. 2010.

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