Software Engineering Institute

Software Supply Chain Risk Management: From Products to Systems of Systems

Robert J. Ellison
Christopher Alberts
Rita Creel
Audrey Dorofee
Carol Woody

December 2010

TECHNICAL NOTE
CMU/SEI-2010-TN-026

CERT® Program

http://www.sei.cmu.edu

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about SEI publications, please visit the library on the SEI website (www.sei.cmu.edu/library).

Table of Contents

List of Figures

List of Tables

Acknowledgments

A portion of the work was funded by the Global Cyber Security Management Branch of the Department of Homeland Security's National Cyber Security Division.

The Acquisition Support Program at the Software Engineering Institute supported an internal supply chain workshop and the writing of this report.

Abstract

Supply chains are usually thought of as manufacturing and delivering physical items, but there are also supply chains associated with the development and operation of a software system. Software supply chain research does not have decades of evidence to draw on, as with physical-item supply chains. Taking a systems perspective on software supply chain risks, this report considers current practices in software supply chain analysis and suggests some foundational practices. The product and supplier selection criteria for system development depend on how a product is used in a system. While many of the criteria for the selection of product suppliers and system development contractors are the same, there is also a significant difference between these kinds of acquisitions. Product development is completed in advance of an acquirer's product and supplier assessment. There is no guarantee that current supplier development practices were used for a specific product. For custom system acquisitions, acquirers can and should actively monitor both contractor and product supply chain risks during development. This report suggests contractor and acquirer activities that support the management of supply chain risks.

1 Introduction

We usually think of supply chains as manufacturing and delivering physical items, but there are also supply chains associated with the development and operation of a software system. Software supply chains include supply chains for physical components, integrated components such as network routers, and software. A supply chain for a commercial software product includes the product development organization and their suppliers. The supply chain for a custom-developed software system can include the prime contractors, subcontractors, and supply chains for the commercial products used.

The growing government reliance on complex software supply chains to deliver military, civil, and intelligence capabilities has increased software assurance concerns. Software supply chain participants have become distributed internationally. This complexity makes it more challenging than ever for acquirers to understand, monitor, and manage supply chain products and processes.

1.1 Hardware and Software Supply Chains

Analysis of hardware supply chains draws on decades of experience and has an established framework for research and analysis. Such a framework cannot exist for software supply chains until a baseline of experience and data exists. Physical and software supply chains share a number of risks such as the business risks associated with a supplier's operations, with delivery on schedule and within costs, and with delivered items meeting specifications. For the shared items, a software supply chain analysis framework can draw on experience with physical supply chains.

There is an important difference between acquisitions for software and those for hardware or integrated components such as routers. A software product is typically delivered as a single item that is then redistributed within an organization. Issues of supply chain integrity apply to that one delivery. Hardware and integrated components involve multiple deliveries of the same item, and supply chain integrity must be verified for each delivery. Hardware specifications can be verified on delivery in most instances, but software functionality cannot. A software component may exhibit undesired behavior when confronted with conditions not considered during development, raising a security concern.

1.2 Software Supply Chain Risk Analysis

A software supply chain can affect all aspects of a delivered system. On-time delivery and costs often get the most attention, but the most serious risks are associated with system assurance. Does the system behave as expected? This report considers software behavior that is associated with security.

Software supply chain risk analysis for security considers three components as shown in Figure 1.

- attack analysis: factors that lead to successful attacks
- supplier: capability to limit product attributes that enable attacks
- acquirer: tradeoff decisions (desired usage and acceptable business risks)
  - business risk assessment—identify attack enablers and possible business risks

- supplier/product assessment in terms of attack enablers and capability of supplier to manage them

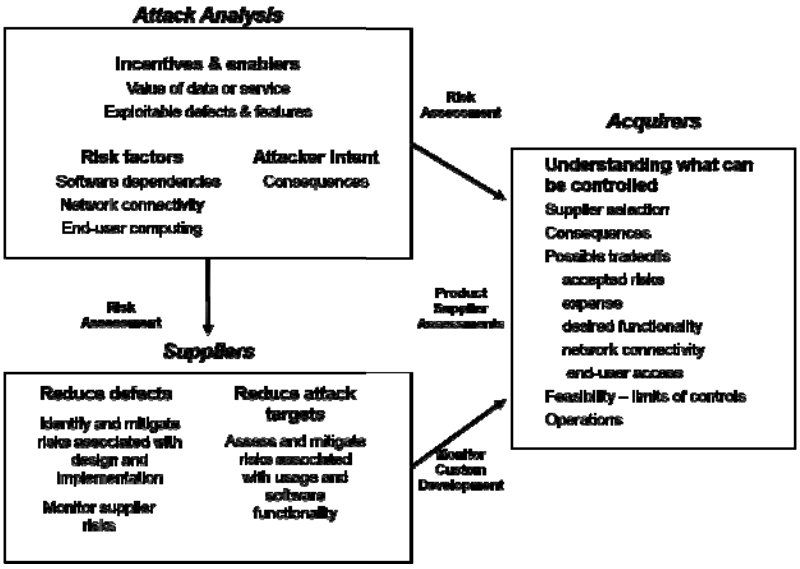

Figure 1: Software Supply Chain Components

1.3 Example: Stuxnet

A recent incident exemplifies the kinds of analysis that support software supply chain risk management (SCRM). In July 2010, malware[1] called Stuxnet targeted specialized industrial control equipment made by Siemens. The malware enabled the attacker to modify how the control system managed a physical system, such as one for water treatment. The designers of Stuxnet did not attempt to compromise control systems directly. Rather Stuxnet infected hundreds of thousands of computers with the objective of compromising a few that were used by system administrators for Siemens control systems, thus enabling the malware to compromise those control systems [Mills 2010].

1.3.1 Attack Analysis

Attack Incentives: Defects

A successful cyber attack typically exploits defects of some kind. The Stuxnet example involves multiple defects, including four previously unknown vulnerabilities in the Windows operating system. Incidents that exploit unknown weaknesses like these are called "zero-day attacks" because any deployed malware or virus detection software would not have been aware of the targeted vulnerabilities.

The Stuxnet attackers had detailed knowledge of Siemens' software and injected their own code that changed how the physical processes were managed by the control system. That this was possible represents a second software and system administration weakness. The configuration of the control system was not sufficiently monitored, so the operational unit was unaware of changes in the process control code.

1 In this report, "malware" refers to software designed with malicious intent.

Connectivity

To avoid detection, the attack did not use the corporate networks to modify the control system software. Instead trusted administrative personnel were used as unknowing transfer agents.

Internet access and defects in Windows or in application software were used to compromise computing resources belonging to the trusted administrators. The transfer to the control software was likely done via USB drives used by system administrators. That such a transfer was possible is also a system weakness. The risks of using the same computing equipment for general use (internet access) and critical administrative activities had either not been considered or not adequately mitigated.

Consequences

The malware could inject code into the control system that affected the physical processes being managed, which could sabotage factory equipment [McGraw 2010b, Richmond 2010]. Analysis of those consequences should consider attacker motivations. In this case, the sophistication of the malware, known as Stuxnet, led to speculations that it was staged by a government or government-backed group, which suggested an attack that targeted specific usage or organizations.

1.3.2 Suppliers

Reduce Targets

Control systems such as the ones targeted by the Stuxnet attack are used to control water, electricity, and nuclear operations. To be adapted to the acquiring organization's requirements, control systems must be extensible. The acquiring organization implements such extensibility by writing the code that controls the physical processes. This kind of extensibility is a frequent target, and in this case, an attacker wrote and installed their own code that adversely changed the behavior of existing control functions. The potential severity of such exploits requires that suppliers give special attention to authorizations, authentications, and auditing for installing such code.

Reduce Defects

Discussions of national cyber threats before Stuxnet had brought up the risks of compromised control systems. In practice, however, mitigating their weaknesses had not been a high priority for control system suppliers, and hence the risk of malware corrupting a configuration had not been considered. A classic tradeoff for suppliers is between costs and extensive failure analysis. In this case, the attack exploited a number of items not controlled by Siemens, such as Windows vulnerabilities and the compromise of administrator computing devices. Given resource constraints, a generic weakness for system development is that developers do not consider all operational software risks, which attackers then try to exploit.

1.3.3 Acquirers

Supplier Selection

While the likelihood of compromising control systems had been the subject of speculative discussions, control system suppliers and products were likely assessed on provided functionality

and on system attributes such as reliability and extensibility. The appearance of Stuxnet, the first instance of a control system exploit, introduces new threats with significant consequences, and it will change selection criteria.

Possible Tradeoffs

There are numerous examples of attacks that exploit compromised end-user computing devices and software. End-user devices such as cell phones can be difficult to control, and software such as web browsers increase attack opportunities. In some instances the advantages of better connectivity outweigh the risks. The connectivity issues in the Stuxnet case were internal to the organization. However, there is increasing network connectivity among the participants in the electrical grid, which raises the risk that the consequences of a supply chain risk at one electrical grid participant will adversely affect others.

Operations

Control system operators should not assume that control system suppliers quickly mitigate these kinds of attacks. A new attack-induced failure could require a partial redesign. In an instance such as the Stuxnet attack, an operational unit would have to tighten system administrative and maintenance procedures to mitigate the risks associated with compromised USB drives.

1.4 Organization of Report

The intended audience for this report includes those developing techniques for software supply chain analysis, those considering the issues that should be addressed, and leading-edge acquirers who seek to identify applicable software supply chain risks that should be addressed.

Section 2 expands on attack analysis. A successful cyber attack typically exploits a mistake. This report concentrates on the inadvertent introduction of exploitable software defects during software system development, which at this time is a significant risk. Section 3 considers supplier capabilities for mitigating supply chain risks. A supplier needs to do a product risk assessment for attributes that might enable an attack. The discussion of desired supplier capabilities describes several development practices. Supplier capability should be measured not by the application of these specific practices but by the application of practices that produce equivalent results. An acquirer must consider both attack enablers and supplier capabilities as described in Section 4. Operational and supplier assessments depend on the type of acquisition, for example, user productivity software, system components, or custom system development. Sections 2 through 4 concentrate on software supply chain risks associated with an initial acquisition. The Stuxnet incident, however, occurred after deployment. Section 5 discusses how software supply risk management should continue into deployment and be integrated into operations. Finally Section 6 provides a summary, and Section 7 discusses possible next steps for this analysis.

2 Attack Analysis

Figure 2 shows the attack analysis part in the software SCRM framework.

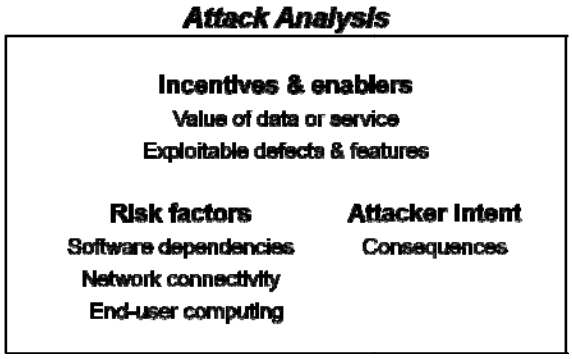

Figure 2: Attack Analysis

2.1 Attack Incentives and Enablers

2.1.1 Defects

Intentionally Created Defects

A successful cyber attack typically exploits a mistake. An intentional insertion of malicious code usually exploits a fault in the system chain. Poor vetting of employees may enable an insider to make unauthorized software changes that create an exploitable defect. Computing software that supports development may be compromised by malware because of poor administration. For example, in 2006 a few Apple video iPods with malware were sold [Apple 2006]. A compromised machine used for quality assurance had introduced the malware.

Many well-understood practices, often drawn from management of physical supply chains, make it harder or riskier for an attacker or supplier to compromise supply chain integrity. The Software Assurance Forum for Excellence in Code (SAFECode) has initiated an effort to address software supply chain integrity [Simpson 2009]. The security of a software exchange between parties can be improved with authentication, signing, and encryption. Having an audited delivery path for computing devices from original equipment manufacturers (OEMs) to the acquirer can mitigate the substitution of counterfeit components. Using vetted employees and strong software configuration management during development reduces the risk of an intentional insertion of malicious code.

Currently, reducing the risk of maliciously inserted code, which is extremely difficult to find, depends on maintaining supply chain integrity. The following statement from the 2007 *Report of the Defense Science Board Task Force on Mission Impact of Foreign Influence on DoD Software* provides a good summary of the issues.

> *The problem of detecting vulnerabilities is deeply complex, and there is no silver bullet on the horizon. Once malicious code has been implanted by a capable adversary, it is unlikely to be detected by subsequent testing. A number of software tools have been developed commercially to test code for vulnerabilities, and these tools have been improving rapidly in*

recent years. Current tools find about one-third of the bugs prior to deployment that are ever found subsequently, and the rate of false positives is about equal to that of true positives. However, it is the opinion of the task force that unless a major breakthrough occurs, it is unlikely that any tool in the foreseeable future will find more than half the suspect code. Moreover, it can be assumed that the adversary has the same tools; therefore, it is likely the malicious code would be constructed to pass undetected by these tools. [DoD 2007]

Inadvertently Created Software Defects

The potential intentional insertion of malicious code at some point in a geographically distributed supply chain presents challenges. However, a much higher risk at this time is an inadvertent exploitable software design and coding error, made by one of the contributors to a supply chain, in the delivered product.

Such exploitable software defects are widespread. On September 22, 2010, Veracode released the second version of their semiannual *State of Software Security Report*, which draws on the analysis of billions of lines of code and thousands of applications [Veracode 2010]. Their overall finding is that most software is very insecure. Regardless of software origin, 58 percent of all applications did not achieve an acceptable security score upon first submission to Veracode for testing. Veracode also tested the software for the 2010 CWE/SANS top 25 most dangerous software errors [MITRE 2010a]. Those weaknesses are well known, easily remedied, and can be identified by commercially available testing tools. Yet 70 percent of the internally developed software and 62 percent of commercially developed software submitted to Veracode did not achieve acceptable security scores when initially tested for those specific weaknesses.

An attack typically tries to put a software system into a state not anticipated by the developers. For example, even behavior well specified in software design can be changed if the system executes attacker-supplied code. Software weaknesses can enable an attacker to change system behavior so as to

- access information not normally available
- create circumstances that lead to termination of a software service (denial of service)
- execute attacker-supplied software

Software defects frequently occur because a design did not consider adverse operational conditions, particularly those associated with increased connectivity and a more complex operational environment. For example, the designers of wireless air pressure monitors for automobiles did not consider input validation, authentication, or encryption. They did not anticipate that those monitors could be accessed by devices external to the automobile that spoof input to cause a false low pressure warning. One supplier's monitor was even damaged [Schwartz 2010].

With an objective to increase developer awareness of common vulnerabilities and attack strategies, as well as how to prevent them, the Common Weakness Enumeration (CWE) describes more than 600 types of software design and coding weaknesses that have enabled cyber attacks [MITRE 2010c]. Table 1 lists some of the CWE/SANS top 25 most dangerous software errors drawn from the CWE. The Common Attack Pattern Enumeration Classification (CAPEC) describes common methods attackers use to exploit vulnerabilities [MITRE 2010b]. In addition,

the U.S. government has sponsored considerable foundational work to create supporting information, standards, and formats for communicating software vulnerability data [NIST 2010b].

Table 1: CWE Weaknesses [MITRE 2010a]

Vulnerability	Potential Consequence
Cross-site scripting—Malware is downloaded as part of a web page.	Information exposure through an error message—provides implementation and configuration details
SQL injection	Buffer access with incorrect length value
Buffer copy without checking size of input	Improper check for unusual or exceptional conditions
Incorrect calculation of buffer size	Improper validation of array index
Operating system command injection—Submitted input is used in parameters in the execution of an external program.	Path traversal—Submitted file name includes "../" which changes directory.

All too often an attack succeeds because a software routine does not properly validate data input. A good example of that is a Structured Query Language (SQL) injection, which has occurred in both custom-developed and commercially supplied software. The cause of SQL injections is well known, and there are several techniques that can be applied to eliminate the vulnerability. Yet it ranked second on the *2010 CWE/SANS Top 25 Most Dangerous Programming Errors* list [MITRE 2010a].

In the following example, information is accessed from a database that uses SQL. Assume an application displays an employee name and salary after a user enters an employee ID. In the typical implementation, an SQL query is constructed by inserting the user-submitted value into an SQL query template. For example, if the user entry is `48943`, the query submitted to the database server might be

```
select Name, Salary from Employees where EmployeeID = 48943
```

A key characteristic of this example is that user input values are incorporated into a string that is interpreted by the server. Such circumstances raise a red flag for knowledgeable developers. In an SQL injection, an attacker's input includes SQL command elements. For this example, consider the input `48983 | (1 = 1)`; in SQL the symbol "|" is the logical OR. This query returns employee records where the `EmployeeId = 48983` or where `1 = 1`, and since the latter is always true, all employee names and salaries are displayed. SQL injections have been used in many of the cyber attacks that gained access to a retailer's database with credit card data.

2.2 Attack Surface

Attackers look for system characteristics that have previously been successfully exploited, which supplies an intuitive notion of attackability. For example, a system with an SQL database may be subject to an SQL injection. An attacker seeks targets with high attackability. Can an acquirer use a similar measure to identify software products with low attackability? Howard proposed focusing on the software features that provide opportunities for attack, which he called the attack surface [Howard 2003]. For example, attacks frequently exploit poor input validation. Hence software components that accept user input are part of the attack surface. An attack surface also includes so-called enablers, which can be components such as an email service, features such as run-time configuration changes, or technologies used to implement a feature. Table 2 shows the elements of an attack surface as used in this report.

Table 2: Attack Surface [Howard 2005]

Targets	data resources or processes desired by attackers (a target could be a web browser, web server, firewall, mail client, database server, etc.)
Enablers	processes and data resources used by attackers to reach a target (e.g., web services, a mail client, XML, JavaScript, or ActiveX[2])
Channels and Protocols	inputs and outputs used by attackers to obtain control over targets
Access Rights	constraints intended to limit the set of actions that can be taken with respect to data items or functionality

For the SQL injection example above, the target is the database, the primary enabler is the use of SQL as the interface between a user-input component and the database, and the channel is the communications link between a browser and the application software that accepts input.

Web browsers by design are extensible, and such extensibility creates large attack surfaces. They can be used for reading email, purchasing, editing documents, and administrating networks. That extensibility is achieved by downloading application-specific executable software (e.g., JavaScript) in addition to data from a web server. Asynchronous JavaScript and XML (AJAX) is a combination of two languages that provides a means of exchanging data with a server and avoiding downloading a full webpage by updating only the parts of a web page that have changed. Attackers want the user to execute their code, and the same features that enable browser behavior at run-time can also be used maliciously to download malware or retrieve confidential user data. HTML5, with over 300 pages of specifications for features like video playback and drag-and-drop, increases the attack opportunities.

PDF file readers have had high attackability. The PDF attack surface includes the language specification of a clickable link that can be used by an attacker to execute an external program. In 2010, most PDF file readers reduced their attack surface by implementing a safe mode in which that capability to execute external programs was turned off by default.

2.3 Attacker Intent

In many instances the objective of an attack that targets an end-user is to access personal information such as credit card data or to use that individual's computing resources to distribute spam email or launch a distributed denial-of-service attack against another specific target. These kinds of attacks target a large community of users.

A specific exploit can also be a step in a larger attack, and how an exploit is used depends on the overall attack objective. An objective in the SQL injection example could be data access. However, compromising site users has occurred as part of a larger attack. As an example, assume a website displays product information and that a web page is constructed based on the user query by retrieving appropriate product information from an SQL database. An SQL injection vulnerability might enable an attacker to modify database entries so that a user query results in the downloading of malware along with product data. A website that does not block such downloads has a cross-site scripting vulnerability. The site is unintentionally distributing externally written scripts.

2 Mechanisms such as JavaScript or ActiveX give the attackers a way to execute their own code.

Compromising an end-user device can also occur in an organization-specific attack. One such attack targeted Google in December 2009 and is illustrated in Figure 3. In this instance, a Chinese team found a new vulnerability in Internet Explorer Version 6 that enabled them to download malware onto the laptop of a Google employee when he accessed an attack-team configured website. In a series of steps, the attackers eventually used that initial browser vulnerability and the user's Google access rights to view sensitive Google source code.

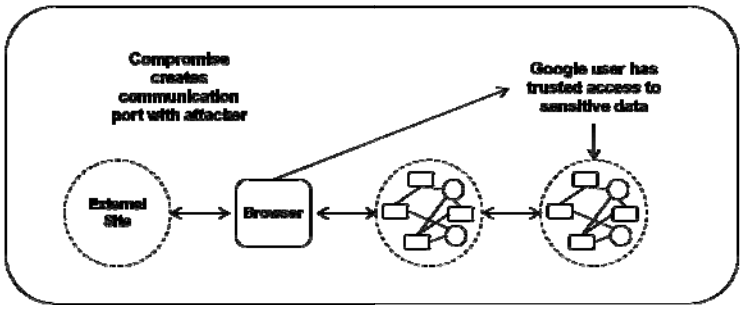

Figure 3: Targeted Attack

A variant of this approach has been used to fraudulently withdraw money from bank accounts, typically accounts for small companies or nonprofit organizations such as school districts. In these cases the first step is to compromise a computer used for bank transactions. The malware is likely installed, as in the Google example, when a web browser accesses a compromised site. The malware in many of these instances has been very sophisticated. The user logs in to create a session authorized for transactions. The malware that had been installed intercepts the bank's web pages and modifies them before relaying them to the user, as shown in Figure 4. The malware typically sends a page that notifies the user of a slight delay in processing. During this so-called delay, the attacker executes multiple transactions to transfer funds to third parties and confirms those transactions in response to bank queries.

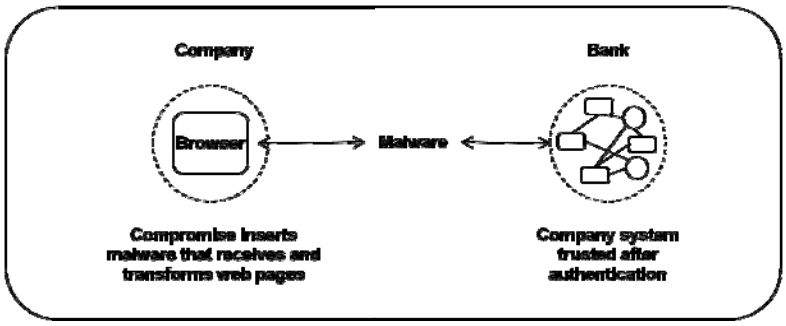

Figure 4: Targeted Banking Attack

2.4 Risk Factors

Several factors affect the occurrence of supply chain risks and the ability of an acquirer to manage those risks. Expanded network connectivity and increased interoperability and dependencies among systems can increase the exposure of a system to adverse conditions. For example, a just-in-time inventory system for a manufacturer or retailer establishes interfaces to supplier systems. A system for a large supplier has interfaces to their purchasers, to their manufacturers, and to organizations that handle transport. A risk for retailers, manufacturers, and suppliers is that one of

the other participating systems has been compromised. The bank in the bank-fraud example has this risk. Identifying possible actions has to start with a design assumption that the other parties may have been compromised. In the bank-fraud example, such an assumption would lead to a design decision to use an independent communications channel for confirmations and not use the potentially compromised communications channel that submitted the transactions.

End-user software has always been a target for attackers. A large user community increases the likelihood of success. When the primary medium of data exchange was a floppy disk, an attacker might have used a Microsoft Word or Excel macro as malware. In 2010 the web is the dominant medium of data exchange, and web pages are used to install malware. Increased end-user connectivity and sophisticated end-user applications increase the likelihood of end-user device compromise. In the bank fraud example, the customer computer was compromised, and in the Google example, one of the company's laptops had been infected by malware. In both instances, the end-user devices had been involved in other computing activities with inadequate controls.

3 Suppliers

Figure 5 shows the supplier's portion of a software SCRM framework.

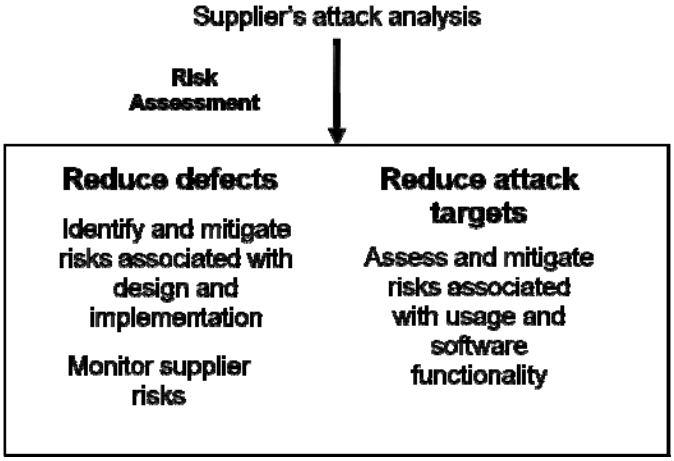

Figure 5: Suppliers

3.1 Reduce Defects

One positive trend is that application security is receiving increased commercial attention. Microsoft's publication of their *Security Development Lifecycle* (SDL) in 2006 served as a starting point for other efforts [Howard 2006]. Today, over 25 large-scale application software security initiatives are underway in organizations as diverse as multinational banks, independent software vendors, the U.S. Air Force, and embedded systems manufacturers. SAFECode, an industry-led nonprofit organization that focuses on the advancement of effective software assurance methods, published a report on secure software development [Simpson 2008]. In 2009 the first version of *The Building Security in Maturity Model (BSIMM)* was published [McGraw 2010a].[3] The Software Assurance Processes and Practices Working Group,[4] operating under the sponsorship of the Department of Homeland Security's National Cyber Security Division, has released several relevant documents, including a *Process Reference Model for Assurance*[5] linked to the Capability Maturity Model® Integration for Development (CMMI®-DEV). In addition, the Open Web Applications Security Project (OWASP) has developed a Software Assurance Maturity

3 BSIMM was created from a survey of nine organizations with active software security initiatives that the authors considered 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.

4 https://buildsecurityin.us-cert.gov/swa/procwg.html

5 https://buildsecurityin.us-cert.gov/swa/downloads/PRM_for_Assurance_to_CMMI.pdf

® Capability Maturity Model and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Model (SAMM) for software security.[6] Finally, the Build Security In (BSI) website[7] contains a growing set of reference materials on software security practices.

While suppliers' software development practices may differ, developers at the leading edge of secure software production agree that secure development practices must be based on threat and risk analysis. A threat refers to an individual or organization that is motivated to compromise a site, while risk is the combination of the compromise's impact and the likelihood that an attacker could exploit the software. A supplier must consider the attack enablers that are applicable to its products. For example, a database supplier should consider both threats and risks. A database often stores financial data and could be the target of threats from organized crime or from insiders. Some threats will have access to the skills and resources necessary for a sophisticated attack. A supplier's software risk analysis has to evaluate their product's resistance to well-resourced attackers. Threat and risk analysis can enable acquirers and suppliers to target a small number of software attributes and focus SCRM resources accordingly.

3.1.1 Threat Modeling

This section focuses on the desired results of software development practices. Threat modeling is a good example of a systematic approach to determining an application's security model during development, and it coordinates efforts among architects and developers to understand threats at design time and throughout construction [McGovern 2010]. Even though the technique is called "threat modeling," the objective is to analyze risks and mitigations rather than to model attacker behavior such as motivations and available resources. As applied to software development, threat modeling is a part of Microsoft's SDL [Howard 2006, Swiderski 2004]. Stephen Lipner has designated it as the most important part of the Microsoft SDL [Geer 2010]. Application of the technique has matured sufficiently that it can be more widely practiced [Steven 2010].

Threat modeling in this report is considered a general-purpose activity that can be applied to systems and workflows in addition to software. Threat modeling incorporates detailed flow analysis. For software, the analysis could consider how one or more data flows or user scenarios among components could be compromised. At the systems level, the first application might be to workflows that involve both people and data. The analysis identifies critical business assets. For software, a detailed walkthrough of a data flow allows consideration of the deployed configuration and expected usage, identification of external dependencies such as required services, analysis of the interfaces to other components (inputs and outputs), and documentation of security assumptions and trust boundaries, such as the security control points. The analysis of data that includes access to an SQL database should either verify the application of known SQL injection mitigations or recommend one. The analysis of usage scenarios supports business decisions by linking threats to business assets. Such a walkthrough can consider adversary motivations, such as the criticality of the data being handled, in addition to the technical risks.

3.1.2 Testing

Increased attention to application software security has influenced security testing practices. All of the organizations initially interviewed for the Building Security In Maturity Model [McGraw

6 http://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model

7 https://buildsecurityin.us-cert.gov/bsi/home.html

2010a] do penetration testing, but there is also increasing use of fuzz testing. Fuzz testing creates malformed data and observes application behavior when such data is accepted. An unexpected application failure due to malformed input is a reliability defect and possibly a security defect. Fuzz testing has been used effectively by attackers to find weaknesses. For example, in 2009 a fuzz-testing tool generated XML-formatted data that revealed an exploitable defect in widely used XML libraries. At Microsoft, fuzz testing finds about 20 to 25 percent of security defects in code not subject to secure coding practices [Howard 2006].

3.2 Reduce Attack Targets

Attack surface analysis is described in Section 2.2. During development, attack surface analysis can focus attention on those software aspects that are of greatest concern for security risks. For each element of a documented attack surface, known weaknesses and attack patterns can be used to mitigate those risks. Well-partitioned code isolates features, reducing the attack surface and the amount of code to be evaluated for threats and vulnerabilities. Such analysis also can identify the attack opportunities that could require additional integration mitigations beyond those provided by a specific product that is used in the system.

3.3 Example of Using Threat Modeling and Attack Surface Analysis

The SQL example is a classic instance of the application of threat modeling and attack surface analysis. A dataflow analysis would consider the effects of the acceptance of invalid data and of identified possible mitigations.

For the bank fraud example, an effective application of threat modeling and attack surface analysis would have identified the security assumption of a trusted user and analyzed the consequences by considering scenarios with compromised clients.

Threat modeling for the Google example could take several forms. The laptop in question was running an older version of Internet Explorer, but the exploits used zero-day vulnerabilities (i.e., they had not been used before). From that perspective the risks of using an older version of IE might be less significant than the effects of zero-day vulnerabilities.

Threat modeling and attack surface analysis could be applied in general to portable devices. The ever-increasing attack surface associated with better connectivity, applications that use that connectivity, and sophisticated malware that targets such connectivity require that threat modeling for internal systems considers scenarios with compromised user computing devices. Such scenarios should not consider how a computing device was compromised but rather should analyze possible effects of such events.

3.4 Reducing Defects and Targets: A Systems Perspective

Vulnerability reports and patches single out a specific product. This may give too much importance to the value of individual supplier and product evaluations. For a software system acquirer, a system perspective is required to provide the necessary context for risk analysis and evaluation of commercial products and custom-developed software components that compose the system.

A software system's supply chain risks are the aggregate of the risks of the system's software components. This includes risks associated with custom software development and integration, with the commercial software products used, and with any use of legacy components.[8] However, vulnerability risk analysis depends on knowing how that software system is used. Usage is derived from the role of the system that incorporates that software component, how the component is used in the system, and the business criticality of the system.

System risk analysis must consider the emergent behavior of the integrated components. Have additional risks been created during integration? No component is risk free. A commercial product might be used in an operational environment subject to greater threats than were considered by its developers. Has component integration considered such circumstances? System integration is frequently outsourced. Hence integration contractors are part of a supply chain, and weaknesses in their integration and SCRM capabilities contribute to the overall set of system supply chain risks. The same risks occur with the use of an internal integration team. Hence supply chain risk analysis should consider both external and internal contributors.

8 In this report, "legacy" refers to previously developed custom software and deployed commercial software.

4 Acquirers

Figure 6 shows an acquirer's portion of a software SCRM framework.

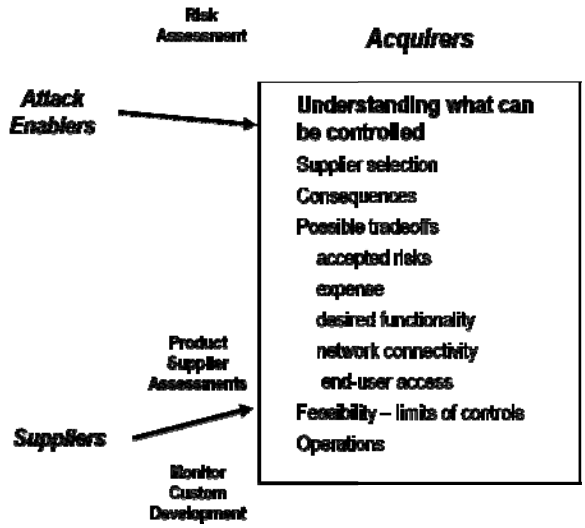

Figure 6: Acquirers

Recent activity has addressed acquirers' management of software supply chain risks. With respect to improving acquirer practices, the National Institute of Standards and Technology recently released a draft interagency report, *Piloting Supply Chain Risk Management Practices for Federal Agency Information Systems* [NIST 2010a]. The Department of Homeland Security (DHS) and Department of Defense's Software Assurance Working Group has published *Software Assurance in Acquisition: Mitigating Risks to the Enterprise*, which contains sample acquisition language and due-diligence questionnaires [Polydys 2008]. The Software Assurance Working Group has also developed top-level recommendations [DHS 2009]. SAFECode has published white papers on integrity controls for supply chains [Simpson 2008, 2009]. In addition, The Open Group has added the Trusted Technology Forum,[9] and OWASP has developed recommended request-for-proposal criteria and procurement language [OWASP 2010]. An effort is underway to build a reference model for assuring software supply chains [Boyson 2009]. Finally, the DHS, in a collaborative effort with the Carnegie Mellon® University's Software Engineering Institute (SEI) and other organizations, has established the Build Security In (BSI) website, featuring a variety of resources for building security into every phase of acquisition and development.[10]

4.1 Understanding What Can and What Should Be Controlled

The most critical acquirer requirement is an understanding of the applicable attack enablers and how they may be managed. Figure 7 shows how the type of acquisition can affect acquirer

9 http://www.opengroup.org/ttf/

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

10 https://buildsecurityin.us-cert.gov/bsi/home.html

controls. Custom-developed software systems provide the acquirer with the ability to monitor and control risks during development. However, systems are increasingly constructed by integrating commercially available software, in which case the only controls might be to accept the risks or not use a specific product. The owner of a system that participates in a system of systems[11] has no control over or knowledge of security risks of the other members [Maier 1996]. Figure 7 shows the effects of two important trends: higher operational risks associated with increased connectivity among systems and reduced acquisition controls with increased use of commercial software as system components.

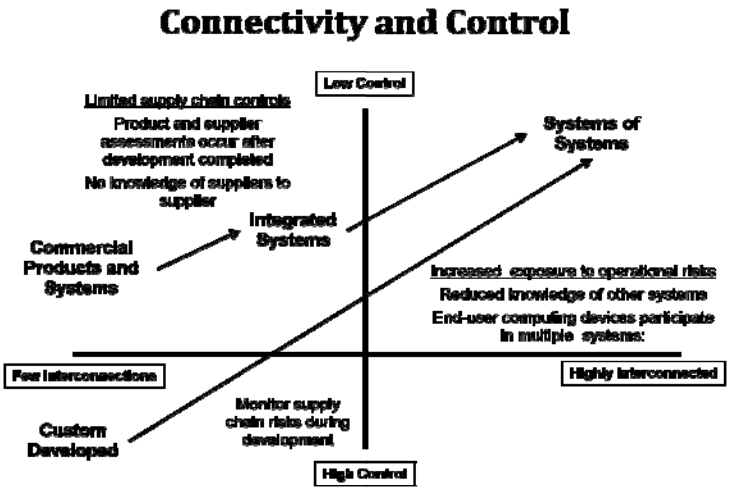

Figure 7: Supply Chain Factors for Acquisitions

4.2 Supplier Selection

For an acquirer, one SCRM objective is to increase assurance that cyber attacks will not compromise operational objectives. Given the security testing failures reported by Veracode [Veracode 2010], deploying software with unknown or unmitigated defects, like the ones in Table 1, adds risk to an already risk-prone scenario. Hence the most critical acquirer SCRM responsibility is to select suppliers that are knowledgeable of the risks discussed in Section 3 and that apply practices, such as those in Section 4, that mitigate those risks. It is also essential that an acquirer understands the limitations of SCRM as discussed in Section 4.5.

Currently, a significant challenge is matching supply chain selection criteria with the assurance level required for the acquirer's business criticality. The use of more stringent supplier selection criteria to reduce supply chain risks can increase the cost of development or reduce the number of commercial products that can be considered. Yet Veracode noted that while banking, insurance, and financial services companies had taken proactive steps and their submitted software had among the best raw security quality scores, their level of security was still not commensurate with their business criticality [Veracode 2010].

11 This report assumes that a system of systems is composed of systems that are independent and useful in their own right, and that they are separately acquired and integrated but maintain a continuing operational existence independent of the system of systems.

4.2.1 Specific Software Products

Most acquiring organizations tend to make single software product acquisitions for end-user productivity software, such as document editors and web browsers, or for integrated systems, such as the Siemens control system or database management systems. Interoperability certainly exists among these products and other systems, but the primary acquisition driver is functionality.

4.2.2 Software Products Incorporated into a System

An acquirer has very little knowledge of the development practices used for a commercial product, and at this time no public certification of a supplier's practices is available. Some organizations do not deploy software products that fail a test suite such as that used by Veracode. At noted in Section 3.1, developers at the leading edge of secure software production agree that secure development practices must be based on threat and risk analysis. An assessment of a product development organization would look for

- a development staff that is knowledgeable in exploitable software weaknesses and well trained in mitigating those risks
- physical, personnel, and industrial security measures
- strong configuration management of development facilities
- careful vetting of employees
- assessment and monitoring of their own suppliers and subcontractors
- attack surface analysis and threat modeling or equivalent practices to identify possible software weaknesses and the strength of mitigations needed given the software's intended operational use
- verification that risk mitigation and remediation actions are sufficient, that testers are knowledgeable of applicable software weaknesses and mitigations, and that those items are incorporated into the test plan

Unacceptable risks identified during a product assessment can lead to a rejection, to the application of operational controls that reduce those risks to an acceptable level, and in some instances to a product revision that has acceptable risks.

Product assessment criteria must reflect the criticality of usage and the level of assurance required. There are no accepted definitions for assurance levels. A general characterization of such levels appears in Table 3. Criteria for high assurance could include independent supplier and product assessments as well as demonstrations of that capability in the development of existing products or systems.

Table 3: Levels of Assurance[12]

High Assurance	Residual risks have been eliminated. There are no known failures.
Medium Assurance	Known vulnerabilities have been addressed. Acquirer understands and accepts any residual risks.
Low Assurance	Designed for low risk and low consequent usage where failure can be tolerated.

4.2.3 System Development and Integration

Acquisition and risk management practices that rely solely on assessments of such products and their suppliers overlook other key sources of risk in today's environment, such as the changing threat landscape and an increasing demand for leading-edge software with risks that are not well understood. These risks should be analyzed from a systems perspective, which captures product usage and consequences associated with supply chain risks.

Quite a few of the criteria for system developers are generalizations of the criteria for product suppliers. However, the application of equivalent practices at a system level can be more demanding. The criteria for development staff is a good example. Staff training for a product's development can concentrate on the development weaknesses appropriate to that supplier's domain and products. Development history usually identifies specific features that require development staff guidance. A product developer normally manages a relatively small and stable set of suppliers. In comparison, an integration contractor or system developer is likely doing multiple one-off efforts across multiple functional domains and with differing sets of applicable software products, suppliers, and subcontractors. It is relatively easy for a product developer to maintain the required expertise given a relatively long product life. New employees can, over time, be brought up the required level. Multiple one-off system developments, on the other hand, require an established base of expertise in a wide spectrum of software weaknesses, threats, and possible mitigations.

An acquirer should assess a contractor's capability to

- analyze software risks associated with the use of commercial products. For example, what access controls are available for the use of runtime customizations? An analysis of proposed usage and a product's attack surface can identify attack opportunities that require mitigations beyond those provided by the product.
- manage risk associated with integration of components that have a lower level of assurance than the desired system assurance level (see Table 3). Examples of this situation include the use of legacy systems or of components designed for a different operational environment.
- perform system-level attack surface analysis and threat modeling or equivalent practices to identify weaknesses related to how software components are used and integrated into the system. For example, proposed usage of a commercial product may encounter operational threats that were not considered by the external developers, or access controls for runtime customizations may not be sufficient for business criticality. An analysis of a product's proposed usage and its attack surface can identify attack opportunities that require mitigations beyond those provided by the product.

12 Drawn from Burton Group presentations and reports.

- maintain a staff that has a broad knowledge of exploitable software weaknesses and their mitigation
- test for applicable system development and integration weaknesses as guided by the system's threat model

4.3 Acquirer Mitigations

An acquirer's options for reducing the occurrence and impacts of software vulnerabilities depend on both the acquisition context and the attack opportunities and motives. Acquirers may act to prevent the insertion of vulnerabilities, recognize and reduce the impacts of latent vulnerabilities, and ensure that the system can be used and sustained in a manner that does not introduce additional supply chain risks.

For custom-developed systems, the most effective acquirer mitigation may be to monitor emerging supply-chain risk during development. While a product with unacceptable risks can be rejected, identification of unacceptable system risks late in a system development life cycle can lead to an expensive redesign or, in the worst case, to the creation of expensive but unused systems.

The development of a threat model and the results of attack surface analysis can guide an acquirer's review of system development activities. A threat model is initially very general and may only note the sensitivity of information or criticality of usage. An acquirer should periodically review threat model development with the supplier to focus attention on critical areas of risk management:

- incorporation of the model in design and implementation, using the threat model to help identify areas of uncertainty that could require additional defensive and recovery measures
- effects of tradeoff, design, and product selection decisions
- threat and risk guidance given to subcontractors and product developers
- product selection criteria based on the threat model and the acquirer's risk profile
- use of the threat model to identify software weaknesses and mitigations that should be part of a test plan
- mitigation of the risks of using leading-edge technology

4.4 Possible Tradeoffs

Developing a system with acceptable supply chain risks always involves tradeoffs. An effective approach for software SCRM must target those risks that affect operational, mission, and business objectives. It must also reflect priorities and resource constraints: not every SCRM practice can be applied to every software component.

Some supply chain tradeoffs are associated with acquirer requirements. A simplified design to reduce cost or speed delivery may not provide adequate mitigations for known operational risks. Products that support end-user runtime customization provide that same capability to an attacker, who could use it to adversely change the behavior of the software. The use of emerging technologies with exploits that are not well understood increases operational risks. System

functionality may have to be changed or a higher risk accepted if mitigation costs for a desired feature are too high or if residual risks for known mitigations are higher than anticipated.

Section 4.2.3 lists general supplier selection criteria for system development. Custom-system development for an acquirer often involves a new use of a software system. A system that requires leading-edge technologies or architectures creates a learning curve for both acquirer and developer. Risks are not fully understood at the start of development. The ability of a contractor to manage the dynamics of risk and communicate those issues to the acquirer is a critical selection criterion.

To help resolve such tradeoffs, a system contractor should be able to

- provide a business justification for security by mapping threats to business assets
- discuss risks and tradeoffs during software development in a quantifiable way [McGovern 2010]

A system contractor's ability to meet the above requirements depends on

- a characterization of an acquirer's risk profile. The acquirer's risk profile, the criticality of usage, the scope of the desired functionality, and the potential integration difficulties should guide the system design and specify product and supplier capability requirements. Adherence to these requirements should be monitored and measured throughout the life cycle.
- analysis that identifies supply chain risks and possible mitigations associated with system integration and the use of externally developed software

Attack surface analysis is valuable in tradeoff discussions for the following reasons:

- A system with more targets, more enablers, more channels, or more generous access rights provides more opportunities to attackers. An acquisition process designed to mitigate supply chain security risks should include requirements for a reduced and documented attack surface.
- An acquirer can compare the security risks and functionality of specific product features.
- The attack surface can also be applied during operations to identify the attack opportunities that could require additional mitigation beyond that provided by the product.

One challenging design tradeoff is between prevention and recovery. For example, which is better for disk drive reliability: drives with better mean-time-failure specifications, or recovery mechanisms that can swap out a faulty drive without affecting operations? Even with recovery, drive quality must be balanced with recovery costs. The use of inexpensive, low-quality drives may require additional drives to provide adequate redundancy, increasing the administrative costs of replacing defective units.

For a number of reasons, recovery is an essential aspect of SCRM. Incorporating extensive prevention mechanisms during development can increase complexity and hence risk. Risk analysis cannot anticipate all possible failure conditions. For example, a commercial product may be used in a high-risk military operational environment with threats not normally encountered in the domestic environment for which the product was designed. Software recovery in these instances often focuses on restoring a software service without an in-depth analysis of the cause, but a successful recovery depends on controlling the propagation of the failure's effects.

4.5 Limitations of Supply Chain Risk Management

An acquirer must be aware of the limitations of SCRM. Total prevention is not feasible because of the sheer number of risks; limited supply chain visibility; uncertainty of product assurance; and evolving nature of threats, usage, and product functionality. Often a system design incorporates what is called defense in depth, or multiple mitigations for a risk, typically at different layers of the architecture. For example, system defenses typically include a combination of network, host, and application controls. System access could be a combination of software and physical controls that implement an organizational policy. Additional mitigations do not necessarily reduce risk. While defense in depth has been demonstrated to protect physical assets, its value for software security has been questioned [Talbot 2010].[13] For complex software systems with multiple mitigations, we often do not understand how interactions among layers or an identified weakness in a layered defense might reduce protection [Talbot 2010]. Such a lack of understanding increases the likelihood during deployment that what are thought to be isolated changes have adverse global effects.

13 Issues raised are consistent with proprietary Burton Group risk management reports based on field observations.

5 Deployment and Operations

One software supply chain challenge during deployment is to ensure that as the threat environment, usage scenarios, requirements, and components evolve, operational risks are continually assessed and mitigated to ensure that operational objectives and assets are not placed at risk. Commercial software components are commonly deployed for five years or longer, but development design decisions and product selections are based on the data available at the time of development. Assessments performed as part of the initial acquisition for a commercial component are valid only at that time.

Some examples of risks that may emerge during deployment include the following:

- New attack techniques and software weaknesses may be discovered.
- Product upgrades that add features or change design can invalidate the results of prior risk assessments and may introduce vulnerabilities.
- Corporate mergers, new subcontractors, or changes in corporate policies, staff training, or software development processes may eliminate expected SCRM practices.
- Product criticality may increase with new or expanded usage.

The transition of a system from development to operations involves several aspects that require careful planning:

- Risks that emerge during deployment in the evolving environment must be regularly assessed. Knowledge of supply chain risks and tradeoff decisions encountered during development must be transferred to the operational unit.
- System maintenance contractors must have the capability to identify and mitigate the emergent supply chain risks as described in Section 3.
- System maintenance contractors must be knowledgeable of potential system weaknesses identified during development and how those weaknesses have been mitigated. System changes made without that knowledge can negate the effectiveness of existing mitigations. This problem is not necessarily resolved by the development contractor continuing in a maintenance role because maintenance may be transferred to a new team.

System development and integration deliverables should include a threat model and a summary of attack surface analysis. The threat model should consider operational usage and threats. Threat modeling supports development during deployment by identifying dependencies, trust boundaries, and key design assumptions. This knowledge is essential to analyzing the impact of changes in usage, threats, and software and to ensure that software upgrades are keeping pace with emerging vulnerabilities. Such a threat model is particularly valuable when responsibility for a system is transferred to a maintenance contractor or an internal unit within an organization. A threat model can also guide requirements for service level agreements.

Software design and integration make assumptions about expected operational behavior. The behavior of complex systems is difficult to model. Developers should not assume that threat modeling is complete or without errors. Operational monitoring should log instances of unexpected behavior, which could indicate invalid design assumptions or an exploited system.

6 Summary

This report considers risks associated with software defects created during development. Such defects are often sufficient to compromise a system. For example, the malware that targeted Siemens' industrial control systems exploited software weaknesses in the Siemens software and in the Windows operating system.

One goal of software SCRM is to assure that cyber attacks will not compromise operational objectives. Achieving this goal requires a focus on reducing the prevalence of software errors that can be exploited to gain unauthorized access, insert malware, or steal or modify critical data and software programs. Robust software development practices can help, but, as noted in the Introduction, software providers have not widely adopted these practices.

Security for application software is getting increased commercial attention, but, for a variety of reasons, it is not widely practiced. A number of efforts are underway to identify criteria for a security evaluation of commercial software products and their suppliers. A general reduction of software defects that could affect product security is essential, but it provides only limited assurance that all the defects that could affect a specific system have been mitigated. Tradeoffs in requirements, component assurance, and costs associated with mitigating supply chain risks that emerge during development should require acquirer concurrence.

In many instances, an acquirer's management of software supply chain risk relies on contractors for system development, integration, and deployment. With increasing system complexity and malware sophistication, system contractors cannot assume that improved product assurance is sufficient. Contracts for system development, integration, or deployment must include requirements for SCRM, and acquirer selection criteria for such contracts must cover supply chain management as well as supply chain risk and threat mitigation capabilities.

7 Next Steps

A key recommendation for acquirers and developers is to use a systematic process that leverages threat and risk analysis for periodic risk identification and mitigation. A solid risk and threat modeling approach will allow acquirers to sort through the plethora of recommendations in the literature and select, apply, and monitor compliance with those recommendations most suited to their own organization or system.

While this report introduced a simple framework that consisted of attack enablers and supplier and acquirer mitigations of those enablers, more work is needed to develop useful tools that facilitate the acquirer's role. The SEI has work in progress that will enable acquirers to effectively identify and mitigate supply chain risks, including

- tools and techniques to clarify the impacts of the software supply chain on operational objectives and systems and to identify the supply chain risks that warrant application of limited resources
- guidance and a framework for selecting and implementing the most promising software SCRM practices
- methods for evaluating and measuring the implementation and effectiveness of SCRM practices and for tracking supply chain risk trends

In addition, the current body of knowledge within the software supply chain domain consists of experiential knowledge reported by a number of sources and codified in various technical reports and presentations within the software assurance and acquisition communities. Supply chain information is often incomplete. There are no demonstrated leading-edge indicators of supply chain assurance that support predictive models of supply chains delivering secure software. At this time, a more rigorous development of the leading indicators of software supply chain security should enable validation or rejection of such measures. It should also enable the construction of predictive models of software supply chain security objectives.

An objective of future work is to build a foundation for longer-term data collection and analysis in support of software assurance and supply chain standards activities. The results are also expected to influence enhancements to training, processes, and regulations related to management of software supply chain risk.

Bibliography

URLs are valid as of the publication date of this document.

[Alberts forthcoming]
Alberts, Christopher; Creel, Rita; Dorofee, Audrey; Ellison, Robert; & Woody, Carol. *A Systemic Approach for Assessing Software Supply-Chain Risk.* Accepted for *44th Hawaii International Conference on System Sciences.* IEEE, forthcoming.

[Allen 2008]
Allen, Julia; Barnum, Sean; Ellison, Robert J.; McGraw, Gary; & Mead, Nancy. *Software Security Engineering: A Guide for Project Managers*. Addison Wesley, 2008.

[Apple 2006]
Apple, Inc. "Small Number of Video iPods Shipped with Windows Virus." *Apple Support.* Apple Inc., 2006. http://www.apple.com/support/windowsvirus/

[Boyson 2009]
Boyson, Sandor; Corsi, Thomas; & Rossman, Hart. *Building a Cyber Supply Chain Assurance Reference Model*. Science Applications International Corporation (SAIC), 2009.

[DHS 2009]
Department of Homeland Security. "Software Supply Chain Risk Management & Due-Diligence." *Software Assurance Pocket Guide Series: Acquisition & Outsourcing, Volume II Version 1.2.* Department of Homeland Security, 2009. https://buildsecurityin.us-cert.gov/swa/downloads/DueDiligenceMWV12_01AM090909.pdf

[DoD 2007]
Department of Defense. *Report of the Defense Science Board Task Force on Mission Impact of Foreign Influence on DoD Software*. Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, 2007.

[Ellison 2010a]
Ellison, Robert & Woody, Carol. "Considering Software Supply-Chain Risks." *CrossTalk 23*, 5 (Sept.-Oct. 2010): 9-12.

[Ellison 2010b]
Ellison, Robert J.; Goodenough, John B.; Weinstock, Charles B.; & Woody, Carol. *Evaluating and Mitigating Software Supply Chain Security Risks* (CMU/SEI-201-TN-016). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tn016.cfm

[Ellison 2010c]
Ellison, Robert. Webinar: *Securing Global Software Supply Chains*. Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/webinars/Securing-Global-Software-Supply-Chains.cfm

[Ellison 2010d]
Ellison, Robert & Woody, Carol. "Supply-Chain Risk Management: Incorporating Security into Software Development." *Proceedings of the 43rd Hawaii International Conference on System Sciences.* Poipu, Kauai, HI, Jan. 2010. IEEE, 2010.

[Geer 2010]
Geer, David. "Are Companies Actually Using Secure Development Life Cycles?" *Computer 43*, 6, (June 2010): 12-16.

[Howard 2003]
Howard, Michael. *Fending Off Future Attacks by Reducing Attack Surface*. Microsoft, 2003. http://msdn.microsoft.com/en-us/library/ms972812.aspx

[Howard 2005]
Howard, M.; Pincus, J.; & Wing, J.M. Ch. 8, "Measuring Relative Attack Surfaces," 109-137. *Computer Security in the 21st Century*, Lee/Shieh/Tygar, 2005. http://www-2.cs.cmu.edu/afs/cs/usr/wing/www/publications/Howard-Wing05.pdf

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

[Maier 1996]
Maier, M. "Architecting Principles for Systems-of-Systems," 567-574. *Proceedings of the Sixth Annual International Symposium, International Council on Systems Engineering*. Boston, MA, July 1996. www.infoed.com/Open/PAPERS/systems.htm

[McGovern 2010]
McGovern, James & Peterson, Gunnar. "10 Quick, Dirty, and Cheap Things to Improve Enterprise Security." *IEEE Security & Privacy 8*, 2 (March-April 2010): 83-85.

[McGraw 2010a]
McGraw, Gary; Chess, Brian; & Migues, Sammy. *The Building Security in Maturity Model, BSIMM2.* http://www.bsi-mm.com/ (2010).

[McGraw 2010b]
McGraw, Gary. "Software [In]security: How to p0wn a Control System with Stuxnet." *InformIT*, September 23, 2010. http://www.informit.com/articles/article.aspx?p=1636983

[Mills 2010]
Mills, Elinor. "Stuxnet: Fact vs. Theory." *CNET News.* October 5, 2010. http://news.cnet.com/8301-27080_3-20018530-245.html

[MITRE 2010a]
The MITRE Corporation. *2010 CWE/SANS Top 25 Most Dangerous Programming Errors.* http://cwe.mitre.org/top25/index.html (2010).

[MITRE 2010b]
The MITRE Corporation. *Common Attack Pattern Enumeration and Classification (CAPEC).* http://capec.mitre.org/ (2010).

[MITRE 2010c]
The MITRE Corporation. *Common Weakness Enumeration.* http://cwe.mitre.org (2010).

[NIST 2010a]
National Institute of Standards and Technology. *Piloting Supply Chain Risk Management Practices for Federal Information Systems* (Draft NIST Interagency Report 7622). NIST, 2010. http://csrc.nist.gov/publications/drafts/nistir-7622/draft-nistir-7622.pdf

[NIST 2010b]
National Institute of Standards and Technology. *The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.1* (Special Publication 800-126 Revision 1, Second Public Draft). NIST, 2010. http://csrc.nist.gov/publications/drafts/800-126-r1/second-public-draft_sp800-126r1-may2010.pdf

[OWASP 2010]
Open Web Applications Security Project. *OWASP RFP-Criteria*. http://www.owasp.org/index.php/OWASP_RFP-Criteria (2010).

[Polydys 2008]
Polydys, Mary Linda & Wisseman, Stan. *Software Assurance in Acquisition: Mitigating Risks to the Enterprise.* The Department of Defense (DOD) and Department of Homeland Security (DHS) Software Assurance (SwA) Acquisition Working Group, 2008. https://buildsecurityin.us-cert.gov/swa/downloads/SwA_in_Acquisition_102208.pdf

[Richmond 2010]
Richmond, Riva. "Malware Hits Computerized Industrial Equipment." *New York Times*, Technology, September 24, 2010. http://bits.blogs.nytimes.com/2010/09/24/malware-hits-computerized-industrial-equipment/?scp=2&sq=control%20systems%20malware&st=cse

[Schwartz 2010]
Schwartz, Mathew J. "Hackers Deflate Auto Tire-Pressure Sensors." *InformationWeek*, Aug. 12, 2010. http://www.informationweek.com/story/showArticle.jhtml?articleID=226700146

[Simpson 2008]
Simpson, Stacy, ed. *Fundamental Practices for Secure Software Development: A Guide to the Most Effective Secure Development Practices in Use Today*. SAFECode, 2008. http://www.safecode.org/publications/SAFECode_Dev_Practices1008.pdf

[Simpson 2009]
Simpson, Stacy, ed. *The Software Supply Chain Integrity Framework: Defining Risks and Responsibilities for Securing Software in the Global Supply Chain*. SAFECode, 2009. http://www.safecode.org/publications/SAFECode_Supply_Chain0709.pdf

[Steven 2010]
Steven, John. "Threat Modeling–Perhaps It's Time." *IEEE Security & Privacy 8*, 3 (May-June 2010): pp. 83-86.

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

[Talbot 2010]
Talbot, E.B.; Frincke, D.; & Bishop, M. "Demythifying Cybersecurity." *IEEE Security & Privacy 8*, 3 (May-June 2010): pp. 56-59.

[Veracode 2010]
Veracode. *State of Software Security Report, Vol. 2*. Veracode Inc., 2010. http://www.veracode.com/reports/index.html

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	December 2010	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Software Supply Chain Risk Management: From Products to Systems of Systems	FA8721-05-C-0003

6. AUTHOR(S)

Robert Ellison, Christopher Alberts, Rita Creel, Audrey Dorofee, Carol Woody

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2010-TN-026

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Supply chains are usually thought of as manufacturing and delivering physical items, but there are also supply chains associated with the development and operation of a software system. Software supply chain research does not have decades of evidence to draw on, as with physical-item supply chains. Taking a systems perspective on software supply chain risks, this report considers current practices in software supply chain analysis and suggests some foundational practices. The product and supplier selection criteria for system development depend on how a product is used in a system. While many of the criteria for the selection of product suppliers and system development contractors are the same, there is also a significant difference between these kinds of acquisitions. Product development is completed in advance of an acquirer's product and supplier assessment. There is no guarantee that current supplier development practices were used for a specific product. For custom system acquisitions, acquirers can and should actively monitor both contractor and product supply chain risks during development. This report suggests contractor and acquirer activities that support the management of supply chain risks.

14. SUBJECT TERMS	15. NUMBER OF PAGES
supply chain, vulnerabilities, secure software, threat modeling, attack surface, supplier assessment, product assessment	45

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL