Considering Software Supply Chain Risks©

Dr. Robert J. Ellison and Dr. Carol Woody
SEI

As outsourcing and commercial product use increase, supply chain risk becomes a growing concern for software acquisitions. Hardware supply chain risks include manufacturing and delivery disruptions and the substitution of counterfeit or substandard components. Software supply chain risks, usually during development, include third-party product tampering or the introduction of exploitable software defects. This article identifies several current practices that can be incorporated in an acquisition to reduce those risks.

Commercial software is not defect-free. There are any common defects such as improper input validation, as defined by the Common Weakness Enumeration (CWE), The MITRE Corporation's list of software weakness types [1]. These weaknesses can be readily exploited by unauthorized parties to alter the security properties and functionality of software for malicious intent. MITRE, in collaboration with the SANS Institute, publishes a yearly list of the Top 25 Most Dangerous Programming Errors [2]. Such defects can be accidentally or intentionally inserted into software, and subsequent acquirers and users have limited ways of finding and correcting these defects to avoid exploitation.

A report by application security company Veracode [3] draws on the analysis of billions of lines of code and thousands of applications that they have analyzed. Their overall finding is that most software is very insecure. Regardless of software origin, 58 percent of all applications submitted for verification did not achieve an acceptable security score for its assurance level upon first submission to Veracode for testing. Table 1 has the results (by source) of software tested against the 2009 CWE/SANS Institute Top 25 list [4]; it shows the percentage of submitted software that passed the security test on the first trial. As 60 to 70 percent of the tested software failed against easily remedied weaknesses, one of Veracode's findings was the lack of developer education and motivation on secure coding.

Software Supply Chain Complexity

There has been extensive analysis of supply chains for delivery of physical material, an analysis based on data collection over decades of practice. The lack of an equivalent base of practice and data collection for software has severely limited the analysis and response to software supply chain risks.

Most supply chains are not a single link between an acquirer and a supplier. A more complex supply chain (such as that shown in Figure 1 on the next page) can involve a combination of internal development, outsourced development, multiple commercial suppliers, and legacy system usage. The composite system inherits the risk of a software assurance (SwA) failure at any point in such a supply chain. The acquirer and the primary supplier have limited visibility of the capabilities of deeply-nested sub-suppliers. Supply chain risks can be reduced but not eliminated. Once software is deployed, residual supply chain risk identification and mitigation become a continuing responsibility for the acquiring organization.

Software supply chain risk considerations must continue in sustainment. An assessment done as part of the initial acquisition for a commercial component is valid only at that time. A commercial software component can easily be deployed for five years or longer. During that period, the following can happen:

- New attack techniques and software weaknesses appear.
- Changes in acquirer usage activate product features with weaknesses that have not been considered in earlier assessments.
- A sequence of product upgrades that add features or change design can invalidate a risk assessment.
- Changes occur in the risk factors used in initial vendor and product assessment (e.g., corporate merger, subcontractors, corporate policies and staff training, or in the corporate software development process).
- Product criticality increases with new or expanded usage.

Mitigating Common Software Weaknesses in the Supply Chain

Addressing the appearance of common software weaknesses introduced in a supply chain requires knowing where to look and what to look for. Discussions of system security often include firewalls, authentication issues (such as password strength), or authorization mechanisms (such as role-based access controls). Application security has often been ignored, in part because of the faulty assumption that firewalls and other perimeter defenses can protect the functional code. The problem is further compounded as application developers without specific security training are typically unaware of the ways their software, while meeting functional requirements, could be compromised. Security software—such as a firewall or a password management component—is usually subject to an independent security assessment that considers the development history as well as the design and operational context. There is no equivalent effort applied to the security of application components.

The pervasiveness of easily remedied weaknesses (as observed by Veracode) provides a simple attack vector that is easily exploited. A first step should be the elimination of the most pervasive common weaknesses, particularly from acquired application software.

There is currently insufficient practice data to identify best practices that could be required of suppliers, but our observation of current practice suggests activities that can improve confidence in a software supply chain [5].

Security for application software is getting increased commercial attention. In 2006, Microsoft established their Security Development Lifecycle (SDL), which served as a starting point for other efforts [6]. Today, more than 25 large-scale application software security initiatives are

Table 1: *CWE/SANS Top 25 Compliance*

Software Source	Acceptable
Outsourced	6%
Open Source	39%
Internally Developed	30%
Commercial	38%

under way in organizations as diverse as multinational banks, independent software vendors, the U.S. Air Force, and embedded systems manufacturers. The Software Assurance Forum for Excellence in Code, an industry-led non-profit organization that focuses on the advancement of effective SwA methods, published a report on secure software development [7]. In 2009, the first version of the Building Security In (BSI) Maturity Model [8] (BSIMM) was published[1]. The Software Assurance Processes and Practices Working Group[2] has released several relevant documents, including [9], which is linked to the Capability Maturity Model Integration for Development. In addition, the Open Web Applications Security Project has developed a Software Assurance Maturity Model for software security [10]. Finally, the BSI website at <https://buildsecurityin.us-cert.gov> contains a growing set of reference materials on software security practices.

The emerging collection of secure development techniques arose from addressing specific software weaknesses. The following section considers three classes of software weaknesses as a way to explain the criticality of software design and coding mistakes.

Common Weaknesses in Applications

Three common weaknesses—cross-site scripting (XSS), SQL injection, and cross-site request forgery (CSRF)—appear in the top four of the 2010 CWE/SANS list. Topping the list is XSS, which can compromise a user's computer when they view a page on what they consider to be a trusted site. Next is SQL injection, an attacker technique that can compromise applications that query databases (e.g., where credit card data has been illegally downloaded). Ranked fourth is CSRF, where an attacker can masquerade as a trusted user of a web server only to upload malicious data to that server.

XSS

Web traffic consists of a mixture of data and script in HTML. With XSS, the attackers objective is to have users retrieve a Web page from your server that contains malicious code, say in JavaScript that the attacker wrote. The user trusts your server, and their browser will execute the malicious code as if it came from you. This vulnerability is a design error that allows the attacker to get their input into your server.

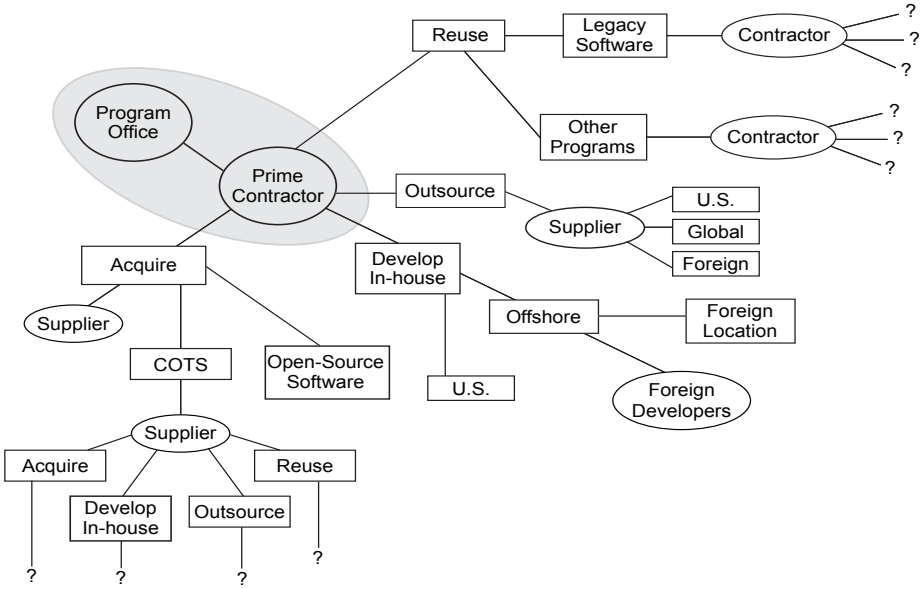
SQL Injection

Weaknesses are often associated with malformed input. The vulnerability risk is high when an application incorporates user input into a service request. Assume we have an application that displays an employee name and salary after a user enters an employee ID. If a user enters *48983*, then a database query is created to retrieve all entries that satisfy the relation *ID = 48983*. An attacker's objective is to see if the input routine will accept values that might provide additional information. The classic SQL injection example would be equivalent to the input of *48983* or (*1 = 1*). If this input is accepted, then the query returns all entries where the *ID = 48983* or where *1 = 1*. As the latter is always true, all employee records are returned.

CSRF

A CSRF is sort of the reverse of an XSS. An attacker compromises a user so that the attacker can masquerade as that user, accessing their Web site and making requests. A CSRF that inserts data—combined with XSS to distribute that data—can lead to extensive and devastating consequences (e.g., XSS worms that spread throughout very large Web sites in a matter of minutes).

Figure 1: *Software Supply Chain*



Emerging Secure Development Practices

Two types of analysis—one focused on understanding and controlling the software attack surface and the other focused on understanding potential threats (threat modeling)—are good examples of SwA practices that can be incorporated early in the development life cycle and that help make supply chain security risk management more tractable. A software attack surface is a way of characterizing potential attack vectors for compromising application code. Threat modeling characterizes which aspects of the attack surface are most at risk for exploitation. These concepts are useful during development, deployment, and system operation. They help guide what information must be gathered and how it can be best used to help prioritize and mitigate (if not eliminate) supply chain security risks.

Attack Surface Analysis

An approach to managing the scope of the software security analysis arose from pragmatic considerations. SDL developer Michael Howard observed that attacks on Windows systems typically exploited a short list of features such as open ports, services running with total access control, dynamically generated Web pages, and weak access controls [11]. 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 (in [12]), with the following dimensions:

- **Targets.** Data resources or processes desired by an attacker; for example, a process could be a Web browser, Web server, firewall, mail client, database server, etc.
- **Enablers.** The other processes and data resources used by an attacker, such as Web services, a 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). These are used by an attacker to obtain control over targets.
- **Access Rights.** Control is subject to constraints imposed by access rights.

An attack surface analysis reduces supply chain security risk in several ways:
- A system with more targets, more enablers, more channels, or more generous access rights provides more opportunities to the attacker. An acquisition process designed to mitigate supply chain security risks should include requirements for a reduced and documented attack surface.
- The use of product features influences the attack surface for that acquirer. The attack surface can define the opportunities for attacks when usage changes.
- It helps to focus attention on the code that is of greatest concern for security risk. If the code is well partitioned so that features are isolated, reducing the attack surface can also reduce the code that has to be evaluated for threats and weaknesses.
- For each element of a documented attack surface, known weaknesses and attack patterns can be used to mitigate risks.
- The attack surface supports deployment, as it helps identify attack opportunities that could require additional mitigation.

Threat Modeling

Threat modeling is a part of Microsoft's SDL [6, 13], but it is a general purpose activity that can easily be incorporated into any development life cycle. Identified as one of 10 low-cost suggestions that improve enterprise security [14], threat modeling:
- Provides a business justification for security by mapping threats to business assets.
- Enables a thoughtful conversation around risk and trade-offs during software development in an objective, quantifiable way.
- Encourages a logical thought process in determining an application's security model.
- Lets architects and developers work together to understand threats at design time and build security in, instead of hoping that the quality assurance team can discover those threats later in the life cycle.

Software Defense Application

Supply chain risks are dangerous for software acquired and utilized by the defense industry. This article examines significant supply chain risks, such as the inadvertent introduction of exploitable software defects during development and third-party product tampering. This article squarely puts responsibility on the acquirer for avoiding supply chain problems, and provides several techniques that will assist defense industry software acquirers in focusing their risk mitigation. These methods will improve software quality, in turn reducing expenses—especially in regards to exploitation recovery and system patching.

- Helps business analysts understand and create traceable security requirements.

The approach used in threat modeling is applicable to other risk assessment methodologies. Data flows or usage scenarios are identified along with critical business assets. A detailed walkthrough 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). The usage scenarios can support business justifications and link threats to the criticality of business assets. Such a walkthrough can consider adversary motivations (such as the criticality of the data being handled), in addition to the technical risks.

Fuzz Testing

Increased attention on secure application software components has influenced security testing practices. All of the organizations contributing to the BSIMM do penetration testing, but there is increasing use of fuzz testing. Fuzz testing creates malformed data and observes application behavior when such data is consumed. An unexpected application failure, due to malformed input, is a reliability bug and possibly a security bug. 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, about 20 to 25 percent of security bugs in code—not subject to secure coding practices—are found via fuzz testing [6].

Using Secure Development Practices in the Software Supply Chain

Let's see how our examples of secure development practices could be applied to the acquisitions of commercial software components. Inputs to that analysis include organization-specific information and available data on vendors and products. The key questions are: *Has the developer considered how the software could be exploited?* and *Has behavior under unexpected or adverse conditions been analyzed?* The evidence to answer those questions can be drawn from coding practices, static code analysis, common weaknesses analysis, attack patterns analysis, threat/vulnerability analysis, software security testing, and dynamic testing.

Techniques such as attack surface analysis, threat modeling, and fuzz testing could play multiple roles in commercial software acquisition.

Assume a commercial component is part of a larger contracted system development acquisition. In this instance, the commercial components are selected by the primary contractor. Supply chain analysis could include examining:
- The attack opportunities the component exposes in terms of features and implementation (component developer, prime contractor, or independently developed).
- The identification and mitigation of risks by the component developer (e.g., supplier fuzz testing, supplier threat modeling [or the equivalent], independent assessment, contractor fuzz testing, and acquirer fuzz testing as part of acceptance and continued for product upgrades during sustainment).
- The criticality of risks for the planned usage (contractor threat modeling as a basis for discussions with the acquirer).
- Risk mitigations (acquirer trade-off decisions with respect to functionality, costs, and acceptable risks based on contractor threat modeling).

Also note that development artifacts should include documented supply chain and threat-modeling analysis provided by the contractor to the acquirer.

Acquirer Responsibilities

Supply chain risks continue during sustainment. A documented attack surface and threat-modeling analysis—provided by a vendor—would influence the acquirer's future responses to changes in usage, threats, or supporting technologies, and should be incorporated into contracting efforts done during sustainment.

While part of the responsibility for

supply chain assurance can be outsourced to a prime contractor, the supply chain risks for individual systems have to be aggregated. For all deployed systems, the responsibility for the aggregation of supply chain risks falls to the acquirer. 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 the criticality of supply chain risk analysis. Software security defects in any of the products or services are a potential supply chain security risk to all SoS participants. A set of one-off approaches for individual system supply chain assurance creates a nearly impossible task for an SoS.

Summary

A software supply chain objective should be to incorporate the identification and mitigation of likely design, coding, and technology-specific weaknesses into the development life cycle. This article provides an analysis of three practices that support that objective. Mitigations of items on a CWE/SANS Top 25 list are usually linked to detailed design or coding practices, but mitigations are also associated with risk analysis, requirements, architecture, and testing. This article—and sources like the BSI Web site—provide a foundation for establishing a full life-cycle context for security improvement.◆

References

1. The MITRE Corporation. *Common Weakness Enumeration*. 17 May 2010 <http://cwe.mitre.org>.
2. The MITRE Corporation. "2010 CWE/SANS Top 25 Most Dangerous Programming Errors." *Common Weakness Enumeration*. 5 Apr. 2010 <http://cwe.mitre.org/top25>.
3. Veracode, Inc. *State of Software Security Report*. Vol. 1. 1 Mar. 2010 <www.veracode.com/reports/index.html>.
4. The MITRE Corporation. "2009 CWE/SANS Top 25 Most Dangerous Programming Errors." *Common Weakness Enumeration* 29 Oct. 2009 <http://cwe.mitre.org/top25/archive/2009/2009_cwe_sans_top25.html>.
5. Ellison, Robert, et al. *Evaluating and Mitigating Software Supply Chain Security Risks*. SEI, Carnegie Mellon University. Technical Note CMU/SEI-2010-TN-016. May 2010 <www.sei.cmu.edu/reports/10tn016.pdf>.
6. Howard, Michael, and Steve Lipner. *The Security Development Lifecycle*. Redmond, WA: Microsoft Press, 2006.
7. Bitz, Gunter, et al. *Fundamental Practices for Secure Software Development: A Guide to the Most Effective Secure Development Practices in Use Today*. 8 Oct. 2008 <www.safecode.org/publications/SAFECode_Dev_Practices1008.pdf >.
8. McGraw, Gary, Brian Chess, and Sammy Migues. *The Building Security In Maturity Model – BSIMM2*. 2010 <www.bsi-mm.com>.
9. DHS. *Build Security In – Software Assurance*. "Process Reference Model for Assurance Mapping To CMMI-DEV V1.2." 23 June 2008 <http://buildsecurityin.us-cert.gov/swa/downloads/PRM_for_Assurance_to_CMMI.pdf>.
10. The Open Web Application Security Project. "Software Assurance Maturity Model." 5 May 2009 <www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model>.
11. Howard, Michael. "Fending Off Future Attacks by Reducing Attack Surface." *Microsoft Developer Network*. 4 Feb. 2003 <http://msdn.microsoft.com/en-us/library/ms972812.aspx>.
12. Howard, Michael, Jon Pincus, and Jeannette M. Wing. *Measuring Relative Attack Surfaces*. 2003 <www.cs.cmu.edu/~wing/publications/Howard-Wing03.pdf>.
13. Swiderski, Frank, and Window Snyder. *Threat Modeling*. Redmond, WA: Microsoft Press, 2004.
14. McGovern, James, and Gunnar Peterson. "10 Quick, Dirty, and Cheap Things to Improve Enterprise Security." *Security & Privacy* 8.2 (Mar.-Apr. 2010): 83-85.

Notes

1. BSIMM was created from a survey of nine organizations with active software security initiatives considered to be the most advanced. The nine organizations were drawn from three sectors: 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, EMC, Google, Microsoft, Qualcomm, and Wells Fargo.
2. The group operates under the sponsorship of the DHS's National Cyber Security Division. See <https://buildsecurityin.us-cert.gov/swa/procwg.html>.

About the Authors

Robert J. Ellison, Ph.D., is a member of the Survivable Systems Engineering Team within the Community Emergency Response Team Program at the SEI, and has served in a number of technical and management roles. Ellison regularly participates in the evaluation of software architectures and contributes from the perspective of security and reliability measures. His research draws on that experience to integrate security issues into the overall architecture design process. Ellison is currently exploring reasoning frameworks development to help architects select and refine design tactics to mitigate the impact of a class of cyber-attacks.

SEI
Carnegie Mellon University
4500 Fifth AVE
Pittsburgh, PA 15213-3890
Phone: (412) 268-7705
Fax: (412) 268-5758
E-mail: ellison@sei.cmu.edu

Carol Woody, Ph.D., is a senior member of the technical staff at the SEI. She leads the acquisition and development practices and metrics team, addressing research in four critical areas for security in software: security requirements, cyber assurance, the software supply chain, and measurement. She is experienced in all aspects of software and systems planning, acquisition, design, development, and implementation in large complex organizations. Woody is a senior member of the Association for Computing Machinery and the IEEE.

SEI
Carnegie Mellon University
4500 Fifth AVE
Pittsburgh, PA 5213-3890
Phone: (412) 268-9137
Fax: (412) 268-5758
E-mail: cwoody@cert.org