SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity

Software Integrity Controls

An Assurance-Based Approach to Minimizing Risks in the Software Supply Chain

June 14, 2010

Editor
Stacy Simpson, SAFECode

Contributors
Diego Baldini, Nokia
Gunter Bitz, SAP AG
David Dillard, Symantec Corporation
Chris Fagan, Microsoft Corporation
Brad Minnis, Juniper Networks, Inc.
Dan Reddy, EMC Corporation

Table of Contents

Introduction

Software assurance is most commonly discussed in the context of preventing software vulnerabilities that arise from unintended coding errors and other quality issues ranging from incomplete requirements to poor implementation. The reduction of vulnerabilities in code is achieved through the application of secure development practices to the software development lifecycle, sometimes referred to as software security engineering.

However, as a more distributed approach to commercial software development has evolved, questions have been raised about what additional product security and commercial risks are introduced in the global software supply chain. One emerging area of concern is software integrity, an example of which is the risk that malicious code could be either intentionally inserted by a threat agent or unintentionally inserted due to poor process controls into a software product as it moves through the global supply chain.

Analyzing this risk in the context of software engineering requires an understanding not only of software security engineering, but also the other essential pillars of software assurance—software integrity and authenticity.

SAFECode defines software assurance as "confidence that software, hardware and services are free from intentional and unintentional vulnerabilities and that the software functions as intended." Achieving this confidence

Three Pillars of Software Assurance

requires software vendors[1] to apply practices and controls to meet three key goals:

Security: Security threats to the software are anticipated and addressed during the software's design, development and testing. This requires a focus on security-relevant code quality aspects (e.g., "free from buffer overflows") and functional requirements (e.g., "passport numbers must be encrypted in the database").

Integrity: Security threats to the software are addressed in the processes used to source software components, create software components and deliver software to customers. These processes contain controls to enhance confidence that the software was not modified without the consent of the supplier.

1. This paper uses both the terms "supplier" and "vendor" to mean an entity that produces software. These terms may be used interchangeably in the real world, and the "vendor" practices listed in this document apply to all software "suppliers." However, in order to be able to describe the relationship between software suppliers without confusion, we are using the term "vendor" throughout the document to identify a specific entity in a supply chain. Thus, in this context, "supplier" refers to an entity that provides software components to the "vendor."

To help others in the industry initiate or improve their own secure development programs, SAFECode has published "Fundamental Practices for Secure Software Development: A Guide to the Most Effective Secure Development Practices in Use Today." Based on an analysis of the individual software assurance efforts of SAFECode members, the paper outlines a core set of secure development practices that can be applied across diverse development environments to improve software security.

The brief and highly actionable paper describes each identified security practice across the software development lifecycle—Requirements, Design, Programming, Testing, Code Handling and Documentation—and offers implementation advice based on the real-world experiences of SAFECode members. These practices are designed to be used in conjunction with the software integrity practices outlined in this paper.

To obtain a free copy of the paper, visit **www.safecode.org.**

Authenticity: The software is not counterfeit and the software supplier provides customers ways to differentiate genuine from counterfeit software.

This paper is focused on examining the software integrity element of software assurance and provides insight into the controls that SAFECode members have identified as effective for minimizing the risk that intentional and unintentional vulnerabilities could be inserted into the software supply chain.

This paper has been developed in conjunction with SAFECode's previously published "Software Supply Chain Integrity Framework," which outlines a taxonomy for the software supply chain and a framework for analyzing and establishing software integrity controls.

The Risks to Software Integrity in a Supply Chain

The risk of an attacker using the supply chain as an attack vector deserves some further examination. Evidence suggests that attackers focus their efforts on social engineering or finding and exploiting existing vulnerabilities in the code, which are usually the result of unintentional coding errors. Thus, experts have concluded that

SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity

a supply chain attack is not the most likely attack vector. Notably, the experiences of leading reputable software companies who work with their suppliers support this finding.

Further, there is growing recognition that 1) there is no one way to defend against every potential vector a motivated attacker may seek to exploit; 2) focusing on the place where software is developed is less useful for improving security than focusing on the process by which software is developed and tested; and 3) there are circumstances when the insertion of malicious code would be almost impossible to detect.

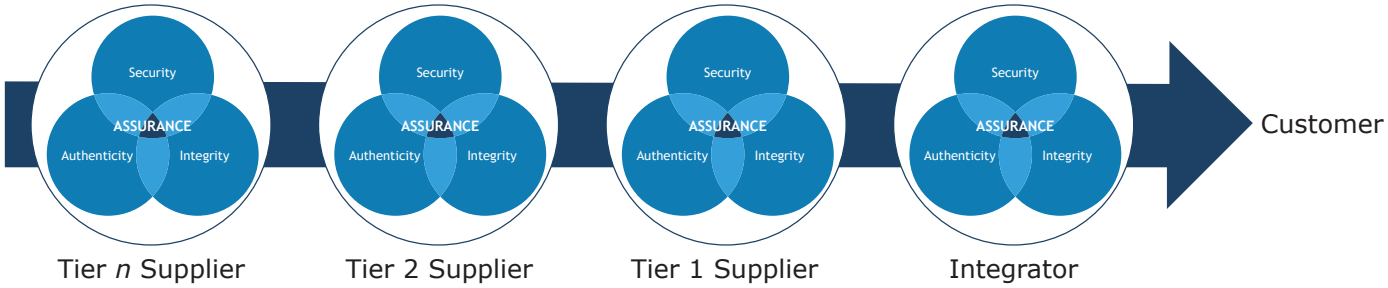
These challenges highlight that a risk from the supply chain could indeed undermine a product's intended function or damage customer trust. Accordingly, major software suppliers take preventative action against any unauthorized changes in the form of software integrity controls. These controls preserve the quality of securely developed code, prevent the inadvertent introduction of vulnerabilities and help to prevent the intentional insertion of malicious code. Vendors leverage these integrity controls to achieve these objectives by addressing the security of the processes used to source, develop and deliver software.

The IT System Supply Chain

The IT system supply chain is a globally distributed and dynamic collection of people, processes and technology. Software is one component of a larger IT solution and each software vendor is only one part of a complex chain of suppliers, systems integrators and ultimate end users. As such, each vendor is only one link of a larger, more complex IT system supply chain.

As a vendor's customer may not be the ultimate end user in the IT system supply chain, it is important to analyze where along the supply chain software security, integrity and authenticity practices and controls can be applied effectively and efficiently.

Each supplier along the IT system supply chain has both an opportunity and a responsibility to apply software assurance practices and controls in order to preserve software integrity,

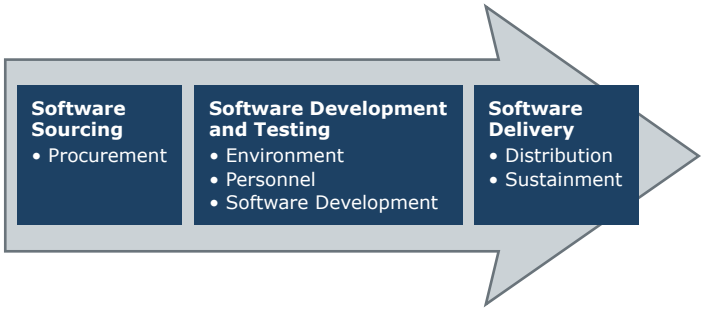


Software Assurance Is a Shared Responsibility In the IT System Supply Chain

security and authenticity within the portion of the software supply chain it controls. Naturally, a vendor has the most direct control over its own internal practices. A vendor's reach into its own suppliers for their software assurance practices and controls may not be as direct.

Within their respective links of the IT systems supply chain, all software vendors control and manage three key lifecycle processes where they can effectively and efficiently implement software assurance practices and controls:

1. Software Sourcing: Vendors select their component and services suppliers, establish the specifications for a supplier's deliverables and have activities to "on-board" software and hardware components and services received from suppliers.
2. Software Development: Vendors build, test, assemble, integrate and package components for delivery.
3. Software Delivery: Vendors deliver the software product to customers and provide ongoing sustainment.



It is within these three processes that effective software security, integrity and authenticity practices and controls must be applied in order to improve the assurance of delivered software. This paper will focus specifically on the software integrity controls that vendors apply to each of these processes.

It should be noted that SAFECode member companies, like industry companies at large, are still sharing information and examining practical and meaningful means of measuring and verifying software assurance in the marketplace. As that work matures, we can expect more consistency in how information about internal processes is asserted and evaluated between trading partners. Thus, while this paper focuses on the practices and controls involved along the supply chain, it was developed with the recognition that more work in this area needs to be done, and it does not attempt to be highly prescriptive with respect to measurement.

Software Integrity Controls

The following sections will detail the software integrity controls that SAFECode has identified as effective for minimizing the risk that vulnerabilities could be intentionally or unintentionally inserted into the software supply chain. This analysis is based on the real-world experiences of SAFECode members. These integrity controls aim to preserve the base level of security in a product achieved through each supplier's

secure development practices by helping to prevent the introduction of vulnerabilities as a product moves along the supply chain.

The controls identified in the following sections are based on the seven basic principles for software integrity outlined in SAFECode's previously published "Software Supply Chain Integrity Framework:"

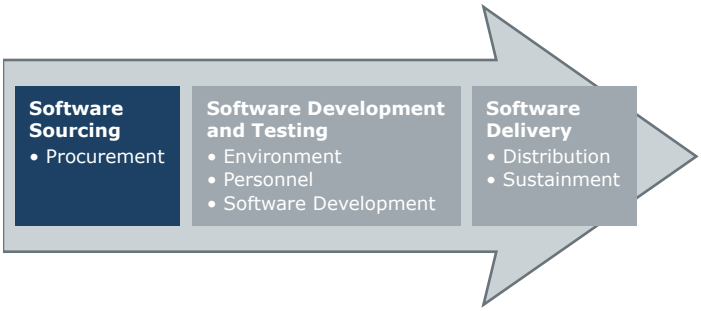
- Chain of Custody
- Least Privilege Access
- Separation of Duties
- Tamper Resistance and Evidence
- Persistent Protection
- Compliance Management
- Code Testing and Verification

These principles support the development of the software integrity controls outlined in this paper and identified by SAFECode as practical, repeatable and auditable.

The software integrity controls described in the following sections do not represent a minimum control list, but rather are designed to be integrated with other security practices and tailored to meet a product's specific risk profile. Furthermore, they are to be integrated into the vendor's software engineering process and performed in conjunction with corporate security functions. These may include physical security, network security, IT infrastructure security and business continuity management.

SAFECode has organized the integrity controls listed in the following sections by the three key lifecycle processes each software vendor has control over—supplier sourcing, product development, and product delivery and sustainment.

Vendor Sourcing Integrity Controls



During the sourcing process vendors establish component specifications, select suppliers of components and services and receive supplied components.

The selection and application of software integrity controls for use during sourcing is a risk-based decision and largely influenced by the nature of the relationship between a vendor and its software component supplier.

There are three types of vendor-supplier relationships: First, "arms length" relationships where vendor A licenses a component from supplier B. Second, work-for-hire relationships where vendor A engages supplier B to provide a software component. Third, work-for-hire relationships where vendor A engages supplier B to provide a staff augmentation service.

Relationships between a vendor and a supplier based on licensing finished components like databases, enterprise resource management systems, or operating systems are examples of arms length relationships. It is incumbent on suppliers that license their software—typically suppliers of commercial-off-the-shelf (COTS) products or Open Source Software (OSS) components— 1) to assure that security threats to the product or component are anticipated and addressed during its design, development and testing; 2) to assure that the processes used to source and create components, and to deliver the product to their customers are secure; and 3) their suppliers provide ways for their customers to differentiate genuine products and components from counterfeit.

In relationships based on work-for-hire, the software delivered by a supplier to a vendor is owned by the vendor. The integrity controls used by the supplier may be the supplier's, the vendor's or any combination thereof. Typically, in staff augmentation engagements the vendor's and supplier's staff work collaboratively on projects that share code libraries, tools and resources, and all project members utilize the same software integrity controls.

In each of the above relationships, the vendor has different degrees of control over the integrity practices and controls used by its supplier. It is this level of control that guides the selection of the software integrity practices and controls necessary to minimize software integrity risks.

The next section describes integrity controls that can be used in a vendor's sourcing process.

Vendor Contractual Integrity Controls

A vendor's engagement with a supplier should be governed by a written agreement, for example a license or a contract. The written agreement must explicitly state the vendor's and supplier's expectations, as well as the consequences of any non-compliance with the terms of the agreement.

Defined Expectations

- Clear language regarding the requirements to be met by the code and the development environment should be set forth during the contracting process. Among other things, this should include commitments to provide security testing, code fixes and warranties about the software development and delivery process used. Overall this helps to set the expectation of delivering a product with integrity.

Ownership and Responsibilities

- Intellectual property ownership and responsibilities for protecting the code and development environment should be clearly articulated.

Vulnerability Response

- In today's world, vendors must push for a more formal understanding of how well their suppliers are equipped with the capability to collect input on vulnerabilities from

researchers, customers or sources and turn around a meaningful impact analysis and appropriate remedies in the short timeframes involved. The fact is that the handling of such vulnerabilities will likely become a joint responsibility in the face of downstream visibility to customers. No one can afford to be surprised about a supplier's potential immaturity in handling these challenges in the middle of a situation. Suppliers provide common terminology for these discussions by using now-default references to well-known specifications like Common Vulnerabilities and Exposures (CVE) and Common Vulnerability Scoring System (the CVSS). Each party should identify contact personnel and review timing and escalation paths as appropriate to be prepared to provide a prompt response.

Security Training

- Another important area for discussion between trading partners is assessing a supplier's capability to effectively train its developers on secure development practices. While it is not necessary to be highly prescriptive about a particular curriculum or certification regime, a company cannot credibly assert that it has a secure development framework or that it follows integrity practices if there is no evidence of any relevant training.

The contracts between companies regarding software have typically been focused on expectations regarding functional performance, defect handling, licensing issues and other challenges like end-of-life support. As concerns about protecting software's integrity have escalated along with reducing the risk of counterfeit components and products, contracts evolved further to address this in language.

New language that specifically addresses the issue of integrity and authenticity of COTS product components from external suppliers that will be included in the ultimate product can also be explored. The language would ask suppliers to self-certify that the supplier's software aligns with security standards and that the supplier's practices align with best practices of industry code security and integrity organizations like SAFECode or its equivalent.

Open Source Software

The use of open source software presents alternative challenges in the context of supply chain integrity.

While in some cases a commercial entity may package and support open source software, other open source software is managed by a community with which a direct relationship cannot be established. In the latter case, the trust and accountability between a vendor and the community supplying software is different. Notably, the contractual terms that vendors establish with commercial suppliers do not apply to community-supplied components as there is no direct supplier with whom to establish an agreement. Existing license terms governing the use of open source software are focused on ensuring that combinations of the software with other software are consistent with the community's expectations. Those license terms may not provide sufficient support for efforts to protect software integrity. Other controls similar to those present in commercial vendor-supplier agreements may need to be implemented for community-supplied software. For instance, as vulnerabilities are visible to anyone and because their exploitability can be readily assessed, open source communities may call for more active vulnerability management and incident handling, and users in the field may request quicker software updates.

As a result, the process used to evaluate and select open source software components deserves consideration. Software vendors analyze the reputation and release engineering practices of the community supporting an open source component to help assess its competence and reliability in dealing with security matters. While the vetting practices will vary depending on the specific product needs and risk profile, means to validate open source packages and their distribution sites need to be adopted and developed, respectively.

A viable integrity control for community open source components is for a vendor to get the source, review it and build it. Validating the quality of open source software needs to happen after acquisition of the code. Vendors may choose to include an open source component or leave it up to the acquirer to obtain and evaluate the component. For vendor-supported OSS, an acquirer can transfer risk to the vendor through appropriate language in their agreement. Otherwise in either case, procedures must be implemented for the inspection of software components for the presence of vulnerabilities and for the assessment of the trustworthiness of the component's distribution site.

In general, a vendor must understand how each of its suppliers handles the open source components that are shipped with its own code.

Vendor Technical Integrity Controls for Suppliers

Secure Transfer

- Delivered code should be transferred securely, using authenticated endpoints and encrypted sessions. Content being delivered should be encrypted for transit. This requires that suppliers use the best available technology, mechanisms and procedures when exchanging deliverables. A secure end-to-end automated process can often strengthen the protection that could be resident in a manual procedure.

Sharing of System and Network Resources

- The digital identities a vendor issues to suppliers to enable access to the vendor's network and resources should be established with strong controls enforced to limit access to only those resources needed to perform the supplier's role.
  - Each resource that is shared should have its own independent assessment as to what authentication and authorization is required. For example, staff access to a vendor's development project requires additional authorization over and above the authorization a staff member receives in order to access a vendor's corporate network.
  - A supplier's access to development assets should expire as soon as it leaves the project. A fail-safe check should also be in place to end all privileges automatically at contract expiration or at another fixed period. A robust procedure is required so that when a supplier's employee leaves the supplier company, the former employee's credentials immediately expire. A combination of automatic disabling and manual notification is best to ensure completeness of privilege removal.

Malware Scanning

- Supplier content to be transmitted to the vendor should be scanned for malware using the most recent malware signature files and more than one commercial scanning engine. While today's malware scanning tools are generally not designed to identify malicious code that is perfectly formed, this standard integrity control should be performed at points of exchange between parties. Depending on the relationship and the practicality of doing so, suppliers should inform recipients of the code as to what scanning has taken place up to the point of transfer.

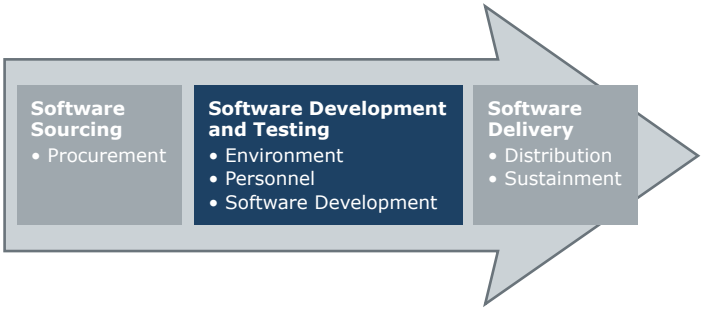
Secure Storage

- Source code for software components and products should be stored securely with need-to-know access controls applied. Code packages that are transferred should be moved to a secure asset repository as soon as practical so that they can be managed more precisely with respect to access privileges.

Code Exchange

- Processes using digitally signed packages and verifiable checksums or hashes should be in place to ensure that received code is complete and authentic. Verifying the digital signatures with validated time stamps of the software packages proves authenticity and establishes that the download or transfer process delivered an intact version of the intended package.

Vendor Software Development Integrity Controls



In software development and testing, software vendors build, assemble, integrate and test software components to finalize them for delivery.

Software vendors have a great deal of experience implementing powerful management, policy and technical controls to achieve sound engineering practices and intellectual property protection. The secure development practices that focus primarily on achieving the "security" circle in the software assurance triad described above become the baseline for internal development.

Within a software vendor's organization, additional software integrity controls may exist within the context of other IT functions such as backup and recovery, business continuity, physical and network security, and configuration management systems. The following are examples of controls employed by SAFECode members:

People Security

- It should be noted that while criminal background checks are often the focus of public debate, in practice SAFECode members have found that they are not as effective as other controls and processes. Focusing on organizational and process controls in conjunction with technology to minimize risks coming from within the company is more efficient and effective. For that reason, many of the following controls to minimize the risk from malicious insiders are based on practices such as the segregation of duties and the use of controlled automated processes.
- It is important that roles, responsibilities and access rights are clearly defined in development processes to achieve a defense-in-depth approach. Development management must be knowledgeable as to who has what access. A team of people with well-planned responsibilities must maintain appropriate operations for guarding code assets while meeting the demands of the global engineering environment.

- In addition to the expected training in secure development practices, there should be training in the secure technical controls used by other integrity practices. Does each organization know how to verify a digital signature with a validated timestamp? Does each organization understand which hash algorithms are best used in a checksum?

Physical Security

- Building security and physical access control should be applied to development locations and code repositories and periodically re-assessed using a risk-based process. Physical security controls should be strong enough to ensure that development assets cannot be accessed by outsiders. Physical protection of source code should go beyond a single layer of building security and include additional distinct physical access controls that limit access to those with a "need to know." For example, additional badge restricted access beyond the normal building access should be required for administrators to access code assets protected in a repository. Physical assets and credentials

Integrity Controls vs. Development Practices

While SAFECode's Development Practices paper describes how to identify and avoid typical coding errors such as buffer overflows, SQL injection, cross site scripting and more, this current work deals with the question of preserving the integrity of an IT product. The integrity practices serve as controls to prevent unauthorized or inadvertent changes to the source code.

Without proper controls, vulnerabilities can be introduced by "good faith" developers. For example, while fixing a problem in their part of the code with dependencies elsewhere, a developer might inadvertently change code while merging it with a related function (e.g., an interface) primarily owned by another. Without proper integrity controls, this change might go undetected and could cause problems elsewhere because nobody was expecting the function to have changed.

A combination of good access controls, testing and peer review of changes could minimize this risk. Thus integrity controls can aim at preserving the well-constructed code for the approved specification while preventing careless or inadvertent changes. Integrity controls throughout the supply chain will also reduce the risk of a malicious attacker being able to change code intentionally or perhaps detect a virus before it spreads into the production environment.

(e.g., keys, badges, security tokens, smartcards, laptops, etc.) loaned to an individual should be retrieved and verified against a list of expected assets as part of a managed termination process.

Network Security

- Network security standards should be established and applied using a risk-based process for the code-related assets. For example, security protections could include intrusion detection or other defensive measures on source code repositories with alerting to appropriate event systems that would alarm during an attack. Session traffic involving source code should be encrypted to acceptable company or applicable industry standards.
- Access to developer workstations should be controlled. For example, workstations can be tied to corporate authentication to ensure that terminated workers are immediately denied further access. Accounts of departing employees and other authorized workers should be properly disabled immediately to allow appropriate review of their work. It is important to disable, and not delete, accounts so that a full forensic analysis is still possible after termination.
- Workstation and virtual machine security should be secured to standards to minimize the opportunity for malicious code to be introduced during the coding process. Developers should have write access to the minimum code necessary to carry out their responsibility. Access to code stored on local machines should also be controlled based on a "need-to-know" and "least-privilege" basis to the extent possible given the goals of the project at hand.

Code Repository Security

- All code-related assets should be housed in source code repositories (also known as configuration management systems or source code control systems), to enable additional attention to security and access control.
- The servers that host the source code repositories should be housed securely. In most major software vendors, these machines are located in data centers with appropriate physical security, hardened server security and business disaster recovery controls. Be mindful that source code is sometimes copied and kept in separate databases after being run through some static code analysis tools. The confidentiality of code files should be protected in all locations. This avoids unauthorized people from seeing the code structure and test results. Combined access to such information might enable them to better target particular code files in a later attack.
- The "out of the box" defaults of any such system must be examined and configured to be secure by default, ideally according to a well-understood standard for a

system holding an acquirer's precious assets such as its customers' personal identifiable information. One objective would be for the system to operate without the risk of allowing exploits through easily inherited system-level root privileges. Many detailed settings such as authentication handling, session variables and external interfaces must be addressed to deliver secure-by-default deployment. A software's default state should promote security. For example, software should run with the least necessary privileges and services that are not widely needed should be disabled by default or only accessible to a small population of users.

- Once enabled as secure by default, that configuration status itself must be protected. As more systems like repositories become compliant with specifications like the emerging Security Content Automation Protocol (SCAP) specifications, the configuration state of the repository and subsequent changes can be expressed and consumed in machine-readable form, offering greater initial and ongoing protection supported by automation.
- Ideally, access to source code repositories should be controlled through the use of corporate identity systems, with strict control maintained over access to any system account. Engineering administrators responsible for managing application repositories should be named users with distinct identities to provide accountability. Administrative practices should observe the separation of duties principle, and elevated permissions should be subject to management approval. For instance, project engineering administrators require a higher level of access to code assets to perform their duties than network security administrators. Other personnel such as IT or Security Operations may have responsibility for base-level configurations and the overall platform profile including security patch levels, etc.
- Within the repositories, access to branches, work areas or code sets must be understood by development management, and access privileges should be granted using the principles of least privilege and need to know.
- Code segments can be tied to specific requirements in a requirements management, enhancement or bug tracking system that allows for cross mapping of functionality to code.
- Change management practices with review and approval paths should be formalized and well understood for code logic and asset changes, repository application and underlying system configuration changes.
- Change logs for all modifications to a product's code assets should be maintained and preserved for future analysis. Logs should provide file names, account name of the person checking in the file,

A company can have source code policy and standards that product engineering teams are expected to meet in the context of protecting source code and product artifacts throughout the product development cycle. For example, these could include detailed corporate expectations regarding the protection of source code repositories and build environments.

Some might be simple requirements, like "Source Code Systems should leverage corporate identity stores for authentication," and perhaps obviously that "no anonymous access can be allowed to a repository." Others are more detailed, such as which particular systems for handling internal request and approval routing for source code repository privileges must be used by each engineering team. Setting up the linkage between source code repositories and the set of build tools is challenging since automation and accountability must be blended. A practical approach is needed such that the sets of tools can be consistent and automated, while still making it known who created and ran the scripted environment that produced a particular build. In addition, build scripts need protection as critical assets. This internal standard also ties into corporate security polices and controls such as the credentialing requirements for personnel and handling of digital identities, a key bridge to best practices around the protection of source code repositories.

An active, ongoing relationship with engineering teams places the internal security team in the best position to effect ongoing improvements to the protection of code throughout its lifecycle. The requirements should not distract by simply attempting to force everyone to use an identical repository, but to set the standard for how a repository should be set up and operated securely. The approach taken in working with engineering teams is to assess the gaps that exist between where a group is today on each item in the standard and to build an improvement plan for closing the gaps as part of a risk-based approach.

check-in time stamp, and the line changes made. They should be kept for a sufficient time in a protected environment to assist with any forensics or ongoing security improvement initiatives.

- A manifest of all code assets contributing to a product, including those developed in-house and by third parties, should be maintained and managed, similar to a Bill of Materials in the manufacturing domain.
- Versions of software assets with their known security characteristics should be tracked in the repository. Change or configuration management should be tracked as well to find the balance between getting the latest patches and updates and having stable, predictable code.

Build Environment Security

- Build environments should be as automated as possible. This minimizes the opportunity for human intervention in the regular build process. However, the "owners" of the build environment should be few. The traceability of actions on build scripts and of access to code files during build should be high.
- Build automation scripts should be treated in a manner similar to other source code assets and checked in to the code repository. This means that changes to the automated build process can be attributed to the person checking in the file.
- Service accounts that run in an automated fashion between source code repositories and build tools should be traceable to individuals with the authority to execute the automated scripts or procedures.

Peer Reviews and Security Testing

One security engineering practice that all SAFECode members use in conjunction with their software integrity controls is security testing. Source code and binary analysis tools, and sometimes manual code review, are performed on code to identify common coding patterns that are known to have been attacked previously. Testing techniques are continually upgraded. Security engineering practices complement software integrity controls because security engineering practices represent an ever-rising threshold against software supply chain vulnerabilities. The testing techniques below are primarily software security engineering practices, not software integrity controls.

Peer Review

- Peer reviews and the manual inspection of code are not often popular given issues of scalability. Automated tools can enable some scalability by collecting and processing more artifacts in preparation for peers performing a focused review. Also, when teams are assigned to work together on code files, an important dynamic is present whereby reviewers can more readily identify code that does not belong within a code set. Focusing peer reviewers on changed code that is scanned again and awaiting

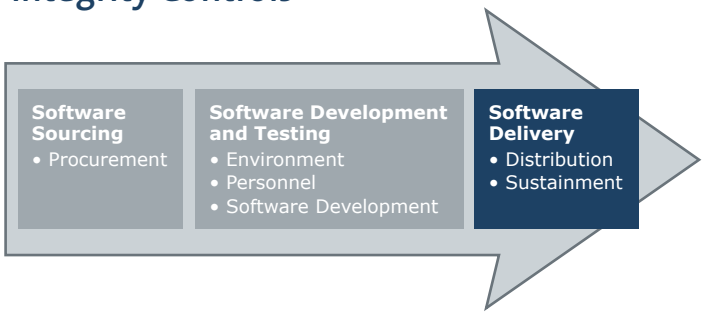
approval during a two-stage check-in to the repository can be an effective approach. Another approach is to couple peer reviews in relation to exercised code paths in the context of overall code coverage. In general, questions about the structure and purpose of sections of code that arise during peer review are more likely to uncover intentional malicious code or inadvertent code errors than automated testing alone.

Testing for Secure Code

- The size of the code base for many software projects today requires automated code review and testing tools. Additional information on secure code testing can be found in SAFECode's "Fundamental Practices for Secure Software Development" paper. Building these tests to run in a repeatable automated manner increases the assurance that they will be performed and analyzed often.
- The list below identifies the most common categories of testing tools used:
  - Static code analysis tools (source code)
  - Network and web application vulnerability scanners (dynamic testing)
  - Binary code analysis tools
  - Malware detection tools (discover backdoors, etc.)
  - Security compliance validation tools (hardening, data protection)
  - Code coverage tools

While security testing is a fundamental part of supply chain security, software vendors recognize that testing alone is not likely to catch malicious code that is intentionally inserted, perfectly crafted and disguised to appear as legitimate. Due to these limitations, software testing must be augmented with the other listed software integrity practices that control access to development assets to more effectively address potential software security risks in this stage of the supply chain.

Vendor Software Delivery Integrity Controls



This stage of the software supply chain covers new product delivery and the delivery of maintenance patches.

It is important to note that while this may be the last stage of the supply chain directly under a software vendor's control, it is not always the final step in the supply chain from the end user's point of view, as software vendors often do not provide their products directly to end-user organizations. In many cases, the software vendor's products are passed to system integrators, resellers and authorized service providers before reaching

the end-user. Thus, as software components leave the supplier, software integrity and authenticity become a shared responsibility between supplier and customer.

Publishing and Dissemination

The controls for product delivery are similar to those for the receipt of code components from software suppliers to the software vendor as described in the *Sourcing* section of this paper. However, additional security needs arise once the software product is complete. These include state-of-the-art anti-malware checks and the availability of a mechanism that provides a way for customers to assure themselves of the integrity of the delivered package.

Malware Scanning

- Products should be scanned for malware using the most recent malware signature files and more than one commercial scanning engine. As mentioned earlier and depending on the nature of the relationship, it may be appropriate to communicate what scanning was done prior to the handover.

Code Signing

- The software vendor's product should be strongly digitally marked with the software vendor's identity in a way that can't be altered, yet may be verified by customers.

Delivery

- A vendor's process for delivering products both online and through distributions using physical and electronic media should be secured. Information on code signing and checksums should be available to customers.

Transfer

- Transfer products in such a way that the receiver can confirm that the product is coming from the software vendor.

Authenticity Controls

For all the work that software vendors do in ensuring they produce a quality product free from vulnerabilities, there remains residual supply chain risk after the product has been released. Millions of customers every year unsuspectingly acquire counterfeit software. According to the Business Software Alliance, over one in five software packages are counterfeit or pirated.[2]

While not a central focus of this paper, authenticity or anti-counterfeiting controls are one of the three essential elements of software assurance and thus are tightly integrated with software integrity controls, especially as

2. Business Software Alliance, "Sixth Annual BSA-IDC Global Software Piracy Study," May 12, 2009.

software is prepared for delivery. Thus, it is important to highlight the key authenticity controls used by software vendors in the software delivery link of the supply chain.

Counterfeit products often look authentic, but they pose serious risks to customers. Counterfeit software cannot be assured to function as intended and often contains malicious code aimed at data destruction or theft. Protecting customers and businesses from the risks of counterfeit software requires both engineering efforts by software vendors and awareness and recognition by acquirers and end users. The risk of counterfeit software can be greatly reduced through purchase from only authorized resellers, careful examination of product packaging and media, and technology to notify users when they may be victims of counterfeit software.

Cryptographic Hashed or Digitally Signed Components

- As mentioned above, digitally signed components or checksum hashes are an essential authenticity control to prove that components are genuine. With any system there are characteristics of the software being shipped that are stable, while there may be other items that vary with particular configuration options as installed. Today the "signing" of an application provides a capability to detect that an application has not been tampered with since the time it was signed. Vendors must find the right balance and offer proof of authenticity for the many predictable aspects of the software.

Notification Technology

- With a variety of distribution channels for software, including online distribution, customers often can't tell that they have a counterfeit product until it is installed on their computer. Vendors can leverage technology to detect certain aspects of the product's integrity and notify the user if the software is deemed to be counterfeit. Sometimes introduced by vendors to prevent license piracy, this technology has evolved into an effective integrity control.

Authentic Verification during Program Execution

- In practice, the integrity of an application can be verified when the application is installed on a computer. Additionally, each time an application runs on a user's computer, similar technology can verify the integrity of the files that make up the application. The hardware and software technology used to verify the claims applications and files make about their validity and integrity is well understood, efficient and broadly available. Software vendors who already make use of this technology have invested in hardware, software, people and process, effectively "code signing" their applications.

- A vendor with the right technology tools can effectively pre-authorize the program execution of only a specific set of applications from a "good" list, effectively blocking any newly spawned code that may not be legitimate.

Product Deployment and Sustainment in the Ecosystem

The software lifecycle extends beyond delivery of the initial software vendor's product and into the product's sustainment or maintenance phase. As a result, patches and hot fixes should be subject to the same software integrity controls as the original code.

It is important that authorized service personnel with ongoing access to genuine parts and proper disposal procedures are involved in the sustainment process. Authorized access should convey that the person actively works for the company providing the service and that service personnel don't have more privileges on the installed environment than those needed to complete the task at hand.

All service transactions should provide evidence that legitimate service personnel did the work, and evidence should be available for audit and protected against tampering.

Secure Configurations

- Whenever possible, software vendors should ship products with a secure configuration being set as the default configuration. Secure configurations for the supplied software should be delivered to the customer along with an outline of the risk implications of the configuration state or choices detailed. The future of broader adoption of machine readable SCAP compliant configurations will strengthen this area's contribution to integrity.

Custom Code Extensions

- Software designed to be integrated and extended to deliver additional functionality creates another link in the supply chain. Assume that the original software and its interfaces were secure, fully functional and delivered with integrity and authenticity. Software components that are added later to extend the functions of an IT System must be also be treated with the same care as originally applied by the internal development and testing of its supplier. Integrators must follow secure development practices as they extend code functionality through the provided secure interfaces. In addition, to continue integrity, their component assets should be cataloged in a repository, access to code restricted based on "need-to-know" and peer reviews implemented. The chain of custody must be preserved with these controls as the sets of products are assembled for the solution to be delivered to the ultimate end customer. Resellers or systems integrators often manage this link in the supply chain.

Table 1: Summary of SAFECode Software Supply Chain Integrity Controls		
Processes		Controls
Software sourcing	Vendor contractual integrity controls	• Defined expectations • Ownership and responsibilities • Vulnerability response • Security training
	Vendor technical integrity controls for suppliers	• Secure transfer • Sharing of system and network resources • Malware scanning • Secure storage • Code exchange
Software development and testing	Technical controls	• People security • Physical security • Network security • Code repository security • Build environment security
	Security testing controls	• Peer review • Testing for secure code
Software delivery and sustainment	Publishing and dissemination controls	• Malware scanning • Code signing • Delivery • Transfer
	Authenticity controls	• Cryptographic hashed or digitally signed components • Notification technology • Authentic verification during program execution
	Product deployment and sustainment controls	• Patching • Secure configurations • Custom code extension

Future Directions

As software integrity remains an emerging discipline, there are a number of areas that SAFECode believes deserve further study and industry collaboration. These include, but are not limited to:

Supplier Management and Communication along the Supply Chain

- The work that the software industry has undertaken to identify and implement secure coding practices, including the findings presented in SAFECode's "Fundamental Practices for Secure Software Development" paper, takes on new implications when examined along the supply chain from one supplier to another. These security practices together with normal quality control concerns could be reexamined in the context of the exchange of software and related information from one supplier to another.

Research on Software Testing

- As discussed previously, automated testing currently is limited in its ability to detect malicious code that is intentionally inserted and well disguised as legitimate. Essentially, today automated testing can only detect malware that use coding patterns that have been seen previously. Increasing the capability of software testing of source and binary code to identify vulnerabilities is an area worthy of future research and development. Additional behavioral analysis of a piece of code might be a promising new approach similar to what is already implemented in some of today's anti-virus detection software.

Authenticity Ease of Use

- While cryptography can be applied with checksums, digital certificates and signatures and validated timestamps, the user experience to verify legitimate software can be confusing and daunting. Users need far easier means of validating authenticity so that they are not primarily focused on clearing their screens of any distractions to get on with the tasks at hand. Since social engineering attacks sometimes count on users dismissing warnings or errors, ongoing work in this area is important.

Authentic Software at Runtime

- How can end-users assure themselves that all software running on their machines is authentic and trustworthy? One promising technology advancement is the Trusted Platform Module (TPM), a hardware component that can be integrated with a signed operating system, signed applications and signed add-ins to provide an end user the assurance at run time that all components are authentic. However, for TPMs to be truly effective, all software must be signed. Some vendors, both community (open source) and proprietary, have taken steps to enable this technology. However, an

industry-wide effort is necessary to achieve this vision as the computers used by end users contain an eclectic collection of software sourced from a vast ecosystem of vendors, suppliers and communities. The Trusted Computing Group (www.trusted-computinggroup.org) is an example of an organization actively addressing this issue.

More Comprehensive Data on Today's Practices and Controls

- While SAFECode has offered the best thinking of its member companies in this important emerging area, the field could be furthered by capturing broader data from a larger segment of information technology vendors about their current or preferred practices so that the overall community is guided by data as continuous improvements are made.

Software Integrity and Cloud Computing

- The impact of cloud computing on the "traditional" view of software supply chain risks, as addressed in this paper, needs to be assessed. Software possesses many of the same characteristics inherent in other forms of intellectual property. As a result, issues associated with jurisdiction, access authorization and compliance need to be assessed for their impact on software integrity controls.

Broader Collaboration with Supply Chain Management Community

- While the well-established and mature supply chain management community is becoming aware of these emerging threats to the IT system supply chain, there is room for greater collaboration around a shared understanding of the challenges, common terminology and existing disciplines that can be leveraged across an even broader community.

Measurement

- SAFECode is currently examining its members' practices on measuring software assurance. As that work evolves, there are sure to be implications for improving the exchange of integrity-related measures among trading suppliers.

Conclusion

SAFECode views software integrity as a fundamental pillar of software assurance. Protecting the integrity of software requires a set of controls that should be implemented alongside secure development and authenticity practices; indeed, integrity preserves and supports security and authenticity across the complexity of a supply chain. However, resources and best practices for identifying and analyzing software integrity controls are not yet widely available, creating challenges for both software vendors and customers.

While a software vendor is only one link in a complex IT solution supply chain and has a limited ability to influence the actions of the other entities along the chain, all software vendors have both the opportunity and responsibility to protect the integrity of the software as it moves through the link they control. This requires the application of software integrity controls to a vendor's software sourcing, development and delivery processes.

SAFECode believes the industry-wide adoption of software integrity controls has the potential to greatly improve customer confidence in IT systems. It has published this collection of best practices, which are based on the lessons its members have learned in their individual implementation of these controls, in an effort to provide guidance to others in the industry. SAFECode encourages the software industry to tailor and adopt these controls, as well as continue further study and analysis on additional practices and controls to improve software supply chain integrity.

Acknowledgments

Brad Arkin, Adobe Systems Incorporated

Eric Baize, EMC Corporation

Matt Coles, EMC Corporation

Robert Dix, Juniper Networks, Inc.

Yuecel Karabulut, SAP AG

Paul Nicholas, Microsoft Corporation

Gary Phillips, Symantec Corporation

Tyson Storch, Microsoft Corporation

Kevin Sullivan, Microsoft Corporation

Janne Uusilehto, Nokia

About SAFECode

The Software Assurance Forum for Excellence in Code (SAFECode) is a non-profit organization exclusively dedicated to increasing trust in information and communications technology products and services through the advancement of effective software assurance methods. SAFECode is a global, industry-led effort to identify and promote best practices for developing and delivering more secure and reliable software, hardware and services. Its members include Adobe Systems Incorporated, EMC Corporation, Juniper Networks, Inc., Microsoft Corp., Nokia, SAP AG and Symantec Corp. For more information, please visit www.safecode.org.

SAFECode
2101 Wilson Boulevard
Suite 1000
Arlington, VA 22201

(p) 703.812.9199
(f) 703.812.9350
(email) stacy@safecode.org
www.safecode.org

SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity