Broadcom and Symantec (A Division of Broadcom) Position Paper on Standards and Guidelines to Enhance Software Supply Chain Security

Issue # 2: *Initial list of secure software development lifecycle standards, best practices, and other guidelines acceptable for the development of software for purchase by the federal government.*

Issue #4 - *Initial minimum requirements for testing software source code*

Author: Stephen Trilling
Title: Senior Technologist
Email: stephen.trilling@broadcom.com

Introduction

Developing a secure end-to-end lifecycle for software development is one of the most important things any software company must do to assure its supply chain is secure. Such a lifecycle requires ongoing evaluation, governance, and updating/improvements when necessary. Key components of any such lifecycle process should include the following:

1. **Securing the "Build Pipeline"**
   - We define the "Build Pipeline" to mean all systems that come in contact with the build process, e.g., source code repositories, engineering workstations, build systems, signing servers, deployment servers (deploying both to on-prem machines and to the cloud).

   - All systems in the "Build Pipeline" must be appropriately secured. Specific recommendations include:
     - All systems should be completely locked down, allowing only those operations specific to each system's function (e.g., build systems can only perform operations necessary to delivering builds and nothing else). This can be achieved via server security solutions (such as Symantec Data Center Security).

     - Web security systems should be deployed to block all outbound web connections from Build Pipeline machines, other than connections to an "approved" list of URLs. Web security systems can also block suspicious incoming web traffic to Build Pipeline machines, e.g., if URLs on the "approved" list get compromised (can be achieved via a solution such as Symantec ProxySG or Symantec Web Security Service).

     - All systems should be monitored for data leakage, particularly code repositories and engineering workstations, via a Data Loss Prevention solution (such as Symantec Data Loss Prevention).

     - Endpoint security software should run in blocking mode on all engineering workstations, to assure that all source and other intellectual property on engineering machines is safeguarded, e.g., intrusion prevention, behavior blocking, reputation-based security, machine-learning based protection, application isolation and control, vulnerability protection (this can be achieved via a solution such as Symantec Endpoint Security).

  - Multi-factor Authentication (MFA) should be required for access to all Build Pipeline systems wherever possible (leveraging a solution such as Symantec VIP).

- In addition to the use of Multi-factor Authentication on Build Pipeline machines (noted just above), access to all Build Pipeline machines should be limited using best practices, e.g., Role-Based Access Control, Least Privilege, etc. See also NIST document Role Based Access Control.

- Best practices should be implemented to protect any secrets associated with the Build Pipeline (e.g., account passwords, API keys, private certificates, etc.). For example, any pages inside the organization containing such secrets should have strong Role-Based Access Control, avoid putting secrets in plain text in any code (e.g., automation code), avoid logging sensitive data in application logs, regularly rotate keys/secrets, see also NIST Special Publication 800-57, Part 1, Revision 5.

- All systems on the Build Pipeline should only be accessible via an "Engineering Network", which is completely segregated from the organization's corporate network.

- All build processes ideally should not have direct access to the Internet if possible. If any such processes do have access to the Internet, such access should be limited only to necessary/authorized sites, as noted in the comments about locking down the Build Pipeline just above.

- The use of Service accounts (e.g., non-human privileged accounts used to run automated processes) should be minimized and carefully audited, e.g., every service account login should logged, including date/time, origin of login; service accounts should be given the minimum privileges; all service accounts should be regularly reviewed to assure they are still needed; per guidelines in NIST Special Publication 800-53.

- Any access to Build Pipeline systems should be logged. All code check-ins/check-outs should be logged at a minimum with the ID of the developer making the changes, time/date of any changes, ID for the code reviewer, and any associated references in the bug tracking system.

- Each code check-in should ideally also be signed by the developer to assure authenticity (e.g., see here).

- All shipping binaries should be digitally signed with a key associated with a root certificate from a Trusted Certificate Authority.

- Signing certificates should not be directly accessible from any build machines, but should be enabled by a purpose-built signing service, for example a Hardware Security Module (HSM). See also NIST White Paper Security Considerations for Code Signing.

- In the event any breach is discovered into the Build Pipeline, there must be clear recovery processes in place to execute to recover from the breach. See also NIST Special Publication 800-61.

- Updates to all in-field software (e.g., patches, product updates, etc.) should be delivered via secure protocols (e.g., HTTPS/TLS). The in-field software products should perform integrity/signature checks on all delivered files to assure the files are valid. This applies to delivering updates both to on-prem and in-the-cloud software products.

2. Additional Considerations for Building Code

- Organizations should consider setting up multiple build systems and then run every new build simultaneously on each of the build systems. Each of the resulting builds produced should then be compared to each other, to assure that they are identical. This will help detect if any one build system has been compromised. Or said another way, a supply chain attacker would need to compromise all build systems simultaneously to impact the shipping software, a much more difficult task than compromising a single build system.

- All builds should be completely reproducible, and the source for each release should be contained in a branch of the source tree that is separate from the trunk.

- Use of all 3rd party/open-source components should be tracked via a Software Composition Analysis tool. All 3rd-party/open-source code and binaries should be kept up to date to the latest version and any such components that have been End-of-Lifed should be removed. In some cases, it may not be practical/possible to remove End-of-Lifed 3rd-party source code – in such cases, the code in question must be maintained by the developer using the component.

- All components/products should be built with all security flags enabled in the compiler.

3. Threat Models

- Threat models should be developed for all software components, as well as for all systems in the Build Pipeline (e.g., all systems in which source code resides or can be accessed, repositories, build systems, etc.).

- Each threat model should be reviewed and approved by at least two independent engineers on the team.

- All code and systems should be reviewed on an ongoing basis against the associated threat model and changes should be made as needed to assure that code/systems do not have structural vulnerabilities.

- Threat models should be updated for major releases or at least annually.

- Threat models should be made available to other internal engineering teams that are picking up or operating any associated software components or systems.

4. Peer reviews

- All code changes should be peer-reviewed by two other engineers prior to check-in.

- The organization should have a centralized Crypto Review Board that assures all shipping code meets company-wide crypto standards. NIST Special Publication 800-175B provides relevant standards.

- The organization should have a centralized Open Source Review Board that assures all shipping open source meets company-wide standards (e.g., shipping latest versions of open-source, removing or supporting any open source that has been End-of-Lifed, etc.)

5. Security testing of all software components

- Code coverage should be a key element of every component's test plan and should be integrated into each build and tracked as part of implementing the test plan. Baseline levels of code coverage should ideally be achieved on all code that is checked in, before new code is committed.

- Security testing should be part of every software component's QA plan and should include the following (see also NIST document "Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework"):

  - Static (linting) and dynamic code analysis should be performed on all code prior to each release using a standard set of company-approved tools. Results of testing should be documented, and all discovered vulnerabilities should be analyzed and fixed.

  - Fuzzing should be performed on all software components to assure that they exhibit expected behavior on all inputs. Results should be documented, and any anomalies/vulnerabilities should be fixed.

  - Penetration testing should be done every 6-24 months depending on potential risk (e.g., cloud products should be pen tested more frequently). All results should be documented, and issues should be fixed.

  - Results of all security testing should be documented, including any CVSS scores and security defects should be fixed and verified (see further details just below in the "Vulnerability Handling" section).

6. Vulnerability Handling

- All known security issues/vulnerabilities should be tracked as product defects in the organization's defect tracking tool, including CVSS score, specific impacts on the component, and any other relevant supporting data. Vulnerability information should only be stored in access-controlled pages in the bug tracking system, given the potential sensitivity.

- Black box testing should be performed to assure that repaired vulnerabilities are truly fixed against all possible attacks.

- The organization should have a central company-wide Product Security Incident Response (PSIRT) team. There should public-facing information (e.g., on a web page) that makes it easy for external researchers to report vulnerabilities in the organization's products. The PSIRT team should work with external researchers to acknowledge and gather information on any reported vulnerabilities, as well as to assure that any reported vulnerability is fixed. Organizations should practice responsible disclosure on all vulnerabilities.

7. Training

- There should ideally be a centralized security team of experts who can help product teams grow their expertise in secure development.

- Individual development teams should be surveyed quarterly to measure compliance against product security goals.

- Every engineer should take 3 hours of security training each quarter, via trainings administered or recommended by the central security team. Attendance should be tracked for all engineers.

- All employees of the organization should also be required to take an annual training around best practices all employees can take to help secure the organization (e.g., how to spot suspicious e-mails, who to reach out to in the organization in case of suspected breach, etc.). This training should include a test at the end of the course to assure understanding of the material, See also NIST Special Publication 800-50.

8. Documentation

- Documentation of security procedures/processes followed should be provided with each shipping software release as much as reasonably possible, without divulging sensitive security information about the product.