5. Guidelines for software integrity chains and provenance

Submitted by:
Jeff Williams, Veteran Application Security Industry Leader
jeff.williams@contrastsecurity.com
https://www.linkedin.com/in/planetlevel/
410-707-1487

We write to encourage NIST to ensure that standards focus on eliminating all three of the following risk areas from the software supply chain, rather than simple integrity confirmations or SBOM reports.

1. known vulnerabilities, including CVEs
2. unknown "zero-day" vulnerabilities, and
3. backdoors and malicious code

While crafting guidance identifying practices that enhance the security of the software supply chain, we encourage NIST to consider the following points:

1. **Consider all four software categories equally risky**. NIST should treat all the software types in the software supply chain listed below (software you write, software you import, software you buy, and software you use) as equally risky.

2. **Expand SBOM to entire application**. NIST should ensure that SBOM covers all software components, not just libraries. Current SBOM standards don't cover all the other components that are part of the application (listed below). Further, SBOM can only be used to reveal known vulnerabilities (1) not unknown vulnerabilities (2) or malicious code (3).

3. **Ensure supply chain integrity**. NIST should ensure that the integrity of all the software in the supply chain (listed below) can be checked during build and deployment steps. Recent attacks, including "dependency confusion" and "typosquatting," have attempted to trick tools and developers into using a malicious library instead of the one intended. NIST can lead the industry here by recommending strong integrity controls from source to production.

4. **Focus on code that runs**. NIST should focus security efforts on code that runs. Despite what you may have read, less than one-third of the running code in a modern application comes from libraries. Vulnerability reports against code that doesn't run are false alarms that undermine security programs.

5. **Don't forget about unknown vulnerabilities**. NIST should create incentives to discover and report previously unknown vulnerabilities across the supply chain. All the code in the software supply chain can and certainly does have unknown or latent vulnerabilities. Discovering these latent vulnerabilities should be just as important as testing custom code.

6. **Don't forget about malicious code**. NIST should create incentives to discover and report malicious code across the supply chain. Malicious code is particularly important for critical software. Current tools and processes do not address this threat at all.

The supply chain standards must be applied equally to all the different types of code in the software supply chain listed below. Attackers will simply shift their efforts to where the defenses are weakest.

A. **Software you write:** This category involves all the source code and configuration "as code" written for a critical system. This category includes code written by external contractors, their partners, and any other sources. There may be many different repositories of custom code, including:
   - Web applications, APIs, serverless
   - Client code, including IOT, mobile, and single page apps
   - Desktop applications
   - Batch jobs
   - Customizations and extensions to existing software
   - Custom security defenses
   - Configuration files, particularly those related to security

B. **Software you import:** Modern software depends on leveraging large numbers of existing libraries, frameworks, and modules that may be open source or otherwise acquired. This software runs with the full privilege of the application using it, so it needs at least the same level of scrutiny. An even higher level of scrutiny is often justified, as the provenance of this software is often indeterminate.
   - Open source and third-party libraries, frameworks, and other modules
   - All transitive dependencies required by included libraries
   - Drivers, plugins, dynamically loading modules, extensions, and other add on software
   - Security defenses

C. **Software you run:** Many software applications and components are acquired from third parties without insight into whether they are secure or not. If these applications and APIs are used for critical purposes or as part of critical applications, they must also be considered critical. We include in this category platform components as well as third party APIs and services, such as SAAS, that provide little insight into their inner workings.
   - Applications and APIs developed by third parties without access to source code
   - Platform runtimes, such as Java or .NET platform and included libraries
   - Application servers, databases, directories, caches, and other repositories
   - Cloud and container images
   - Third party web APIs, web services, microservices, RPC services
   - External services, particularly those related to security

D. **Software you build with:** Finally, the software we use to build and operate critical applications and APIs is itself critical. As we have seen in recent breaches, a compromise in any of the tools used to build, package, test, and deploy code could enable an attacker to completely take over an application.
   - IDEs
   - Build servers
   - CI/CD tools
   - Deployment tools
   - Performance tools
   - Testing tools and frameworks
   - System administration, monitoring, and configuration tools
   - Any other tools running on development machines