PUBLIC SUBMISSION

As of: 2/28/22 10:48 AM
Received: February 24, 2022
Status: Pending_Post
Tracking No. l01-1act-udke
Comments Due: April 25, 2022
Submission Type: Web

Docket: NIST-2022-0001
Evaluating and Improving NIST Cybersecurity Resources: The Cybersecurity Framework and Cybersecurity Supply Chain Risk Management

Comment On: NIST-2022-0001-0001
RFI-2022-03642

Document: NIST-2022-0001-DRAFT-0004
Comment on FR Doc # N/A

Submitter Information

Email: Rossa Young
Organization: Erudite Candor LLC

General Comment

See attached file(s)

Attachments

SupplyChain

Let's talk about 6 important steps that you can do right now to secure your organization against supply chain attacks.

1. You need to **centralize your software code repository**. You do not want to have to check github.com, 100 private gitlab repositories, 25 bitbuckets, and 20 codecommit locations. You want everything to be in one Software Code Repository. Thus if someone says do you have XYZ software in your organization, you can say let me do a quick search to see if any software repositories return a result. You should also make sure that your software code repository ties back to ownership. Who owns each software repository and is responsible for maintaining the code. This person who hopefully is still in your company can make the call to remove software that is outdated.
2. You need to **centralize your artifact repository**. Solutions such as Artifactory or Sonatype Nexus provide a location for you to host all of your binaries, containers, and executables across your organization. Remember if developers can go externally to download software whenever they want vs using approved software from an approved source, then it's just a waiting game till a developer finds malware or cryptominers. Make it easy to say here's a mirror of every NPM, Java, RedHat library within our own network on your artifact repository. This allows you to gain important insights from a security perspective. When a piece of software is identified as bad you can say not only do we have this library within our artifact repository, but we also know that it has been downloaded 1,000 times by developers. Additionally, if you want to remove it for the organization after you are sure it won't break the software of others, this is one place you can do that.
3. You should **scan open source software for malware**. If someone you didn't know emailed you an executable would you just double click it? Hopefully not. If you thought there was an important business reason to see the executable then you should at least scan it for malware. Truth is most software organizations don't do this. As a CISO you should see if the artifact repository solution used by your organization has two types of malware scanning. First, Does it check for malware before adding open source software to it's holdings? Second, Does it have a recurring scan to identify if malware is within it's holdings when malware signatures get updated by the Antivirus Software? This will minimize crypto miners and other harmful malware from staying in your organization. Here's one way you can build a secure pipeline for malware scanning. Take open source software that your developers want to use and put it in a public S3 bucket with read only access. Then use virus total to perform a URL scan on your S3 bucket. This allows you to scan open source software with 50+ different antiviruses. Since AV software is known to have false positives you might set a threshold that says if 3 or more of the 50 antivirus vendors flag this as malware then we will not allow this software to enter the company.
4. You need to **scan software for vulnerabilities and vendor support**. One of the biggest problems today is developers leverage software with known vulnerabilities. This can be because there's issues with running the latest version of software, there's no patch for xyz vulnerability since it just came out, or we didn't know newer software exists. Most organizations will purchase software called Software Composition Analysis Software. Things like Blackduck, Snyk, or GitHub's Dependabot will scan your software and say you are on version 1.3. Which is known to contain 2 critical vulnerabilities and 5 highs. You should update to version 2.1 or newer. This is helpful information. Developers can put this into a continuous integration pipeline with tools like Jenkins or GitHub actions that check all software for vulnerabilities. If you see a high or critical vulnerability then please break the software build and don't release it. Guess

what, if Log4J gets a critical vulnerability or CVE score of > 9 assigned to it, then your developers get timely feedback that they need to update as soon as possible since they can't release their build. The one thing to also understand here is that just because there isn't a public CVE doesn't mean the software is good. There's a lot of software that doesn't have support from a vendor. So developers should actually use tools like Maven Versions if you are writing java code to detect if your software is on the latest version of software. This is super important. Let's say a software company has a huge bug in their software that gets patched. In the release notes the company says bug fixes. Well you don't know if the bug is a software vulnerability, a memory leak which can crash your app, or something else. Plus new software usually has new features which can help your organization. So please stay current because software ages like milk not like wine. Tools like Maven Versions will tell you how far every java library that you declare in your software bill of materials is from the latest version from the vendor.

5. **Run a Web Application Firewall** to quickly patch your organization. Let's say you have 100 development teams that run a piece of vulnerable software such as Log4J. Your cyber security team has to track the status of 100 teams to identify when they are all patched to ensure the organization isn't vulnerable. Well then you are only safe when your slowest development team is patched. This could be weeks or months. Enter the biggest benefit of a Web Application Firewall or WAF. If every web application sits behind a WAF, then you might only have to make a change to the WAF to block Log4J on all applications. This means you can secure the organization quickly and give the developers some breathing room when the fix takes longer to patch than hackers can exploit.
6. **Run a Runtime Application Security Protection tool**. One of the best features of a RASP is it can inspect software for application libraries. Remember this: Vulnerability Management Tools like Qualys and Nessus will miss finding custom applications containing Apache Struts vulnerabilities. Now it's true that Software Composition Analysis and Container Scanning tools can find application libraries. There's things those tools don't have that a RASP does. They don't tell you if the vulnerable library is actually in production. You only know if the vulnerability is in your code. RASP is used on Production boxes so now you know. You can quickly query RASP technologies to obtain an accurate software inventory for application libraries used on the same box as a RASP agent. RASP actually protects software, unlike SCA/container scans since it has an in-built WAF. Financially RASP also has another amazing feature. RASP can actually look into binaries and tell you what libraries they are running. This is super helpful when looking at Vendor Software. Let's say you host financial software on your own servers. You can install a RASP agent on the software and look at application libraries the financial software is running. You could find out if Log4J or OpenSSL was being used by these vendors and then reach out to the vendors and ask for a patch. If you didn't have RASP, you would probably be flying blind unless the vendor publicly exposes they are vulnerable or provides a software bill of material.

So now that you know the important steps you can take to defend your organization let's talk a little about where the industry is headed. There's a new industry collaboration called the Supply Chain Levels for Software Artifacts, or SLSA (salsa). https://slsa.dev/ This organization is led by a vendor neutral steering committee from Citigroup, Cloud Native Computing Foundation, Google, The Linux Foundation, Intel, VMWare, Datadog, and Chain Guard. They are creating levels of security for organizations to follow. Rather than just running untrusted software we need to add some security.

- Step 1: Lets introduce a software build process that is fully automated and generates provenance. Provenance is metadata about how the software artifacts were built, including the build process, the source code, and it's dependencies.
- Step 2: Require using version control and generate authenticated provenance. We can think of a certificate authority that ensures that a website is who it says it is. We need signed software from vendors that is unique and trusted.
- Step 3: We need an accreditation process whereby auditors certify that platforms meet requirements. Similar to having a SOC 2 Type 2 report or ISO 27000 an outside auditor can show that a company actually does something after reviewing evidence.
- Step 4: Requiring two-person review of all changes and a reproducible build process. Similar to moving money for a bank, a two-person review is an industry best practice for catching mistakes and bad behavior. It limits one person's ability to cause harm or fraud. We need more checks and balances for software.