Rezilion

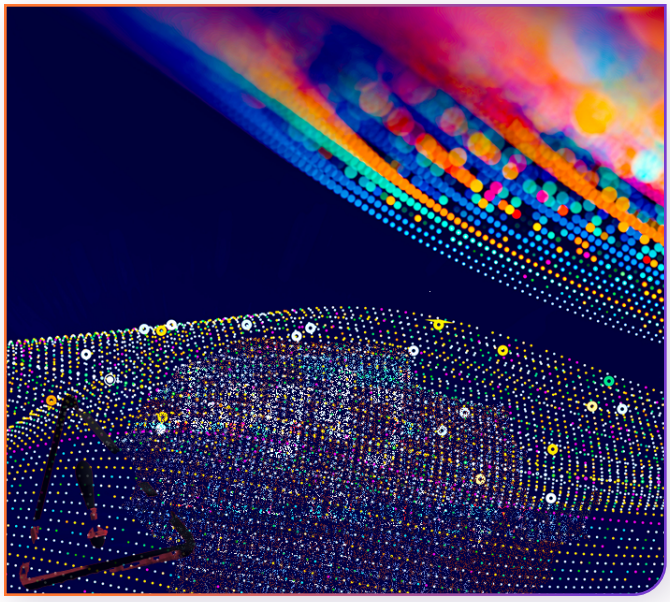
Using Dynamic SBOM to Discover Log4Shell

YOTAM PERKAL AND BAKSHEESH SINGH GHUMAN

Many organizations globally are painfully familiar with the Log4Shell vulnerability (CVE-2021-44228). The affected Log4j library is widespread, yet not always easy to detect, especially in a production setting. It is critical to maintain a Software Bill of Materials (SBOM) as a significant first step to gain visibility of potentially vulnerable applications.

However, what is also clear, is that an SBOM (as it is perceived today) is only a first step. In order for security teams to be able to effectively track software components across the supply chain and use the SBOM as a tool for triaging Log4Shell like vulnerabilities (reducing time-to-detect and time-to-patch), an SBOM must:

- **Be machine readable** in order to allow for automation. Some positive strides are being made with standards such as SPDX and CycloneDX.
- **Contain security context,** an ingredient list of the software is not enough. Ideally, an SBOM should be able to report if the package is actively maintained, if it is in an end-of-life status, if it has a critical vulnerability, if it is exploitable in the context of the environment in which it is running, etc. Adoption of standards such as Vex (Vulnerability Exploitability eXchange) could help drive this change.
- **Be dynamic,** not all packages in the development or CI environment find their way to production, even those that are not necessarily used. A dynamic SBOM should have the ability to identify which components are actually being used (loaded to memory) and which are not, thus providing valuable context for tasks such as code debloat, risk assessment, or exploitability evaluation.
- **Be continuous,** an SBOM should not only be generated once at a given point-in-time or at a specific point in the Software Development Life Cycle (SDLC) pipeline. Instead, it should be a continuous, ongoing process.
- **Be searchable** in order to find out if known vulnerabilities and components are present in your environment.



Our latest research, which examines the prevalence of Log4j related vulnerabilities four months after the first publication of Log4Shell, demonstrates that what security teams are doing to tackle the Log4Shell beast is not always enough. The research finds over 90,000 publicly facing servers are still running obsolete versions of Log4j, and showed that the time-to-patch for some of the vulnerable applications exceeded 100 days. To date, over 33% of the Log4j downloads from Maven Central are of vulnerable versions.

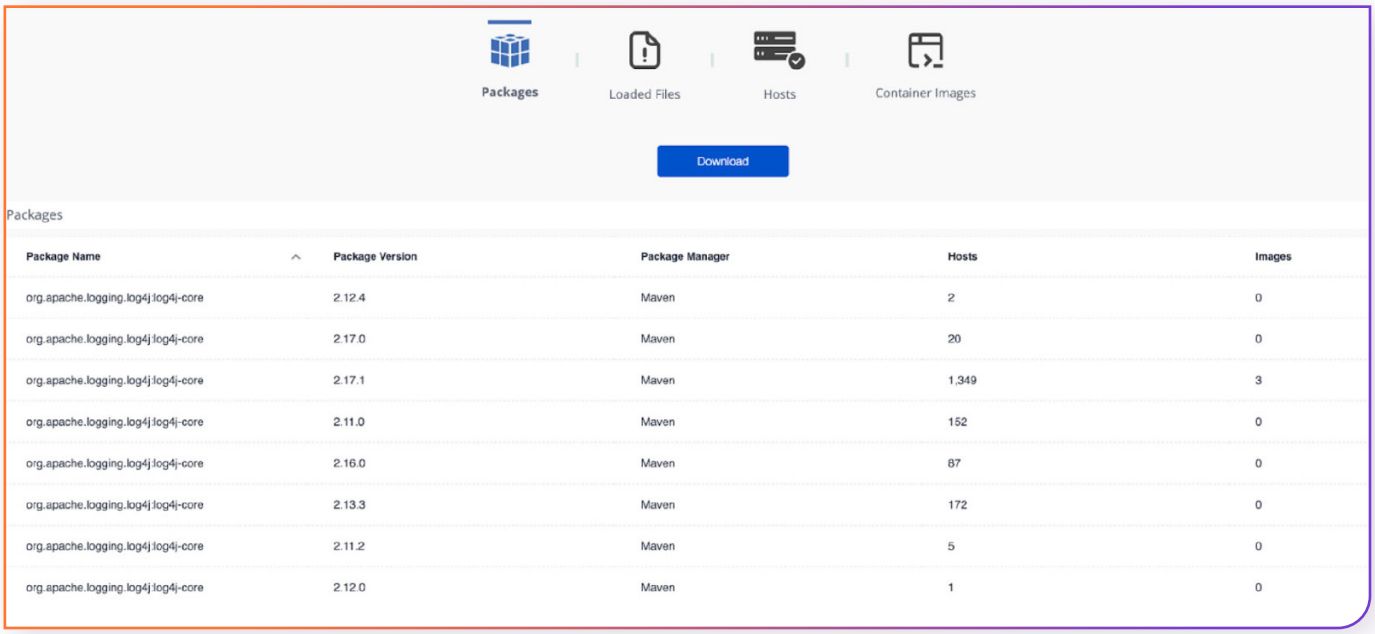
These findings could be explained by the following challenges:

- **First,** the biggest challenge is finding your Log4j vulnerabilities. Log4j, due to its nesting nature, is hard to find. If you can't see it, you can't patch it or mitigate it.
- **Second,** the pervasive nature of Log4j makes it a large problem over an even larger attack surface – CI/CD pipeline, on-prem server, or cloud workload, network, etc.
- **Third,** a significant amount of the vulnerable attack surface stems from third party code (whether commercial or open source). The visibility into the software components of these third-party applications is often limited.
- **Finally,** some organizations lack the resources and/or the proper vulnerability management processes in place to combat this threat.

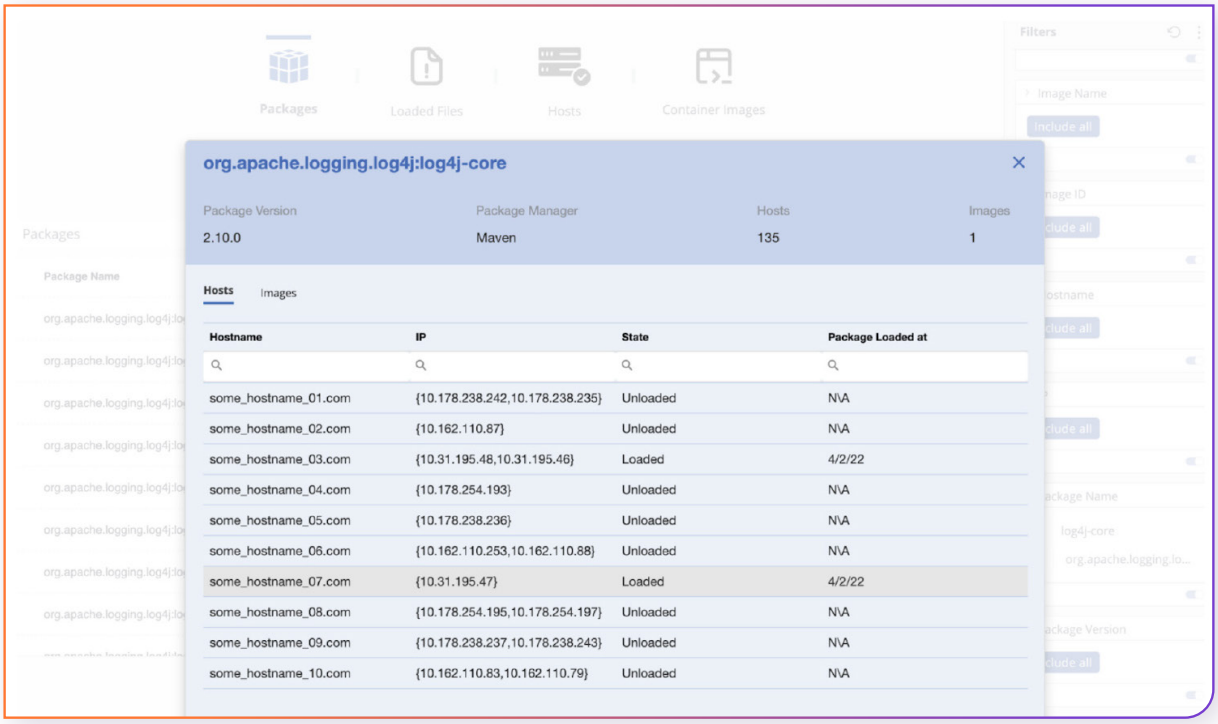
A dynamic SBOM can help customers manage their Log4j challenges in three easy steps. Here's how:

STEP 1: **Create a Dynamic SBOM**

Unlike traditional scanning methodologies that are point-in-time or scheduled scans, dynamic SBOMs are continuously updating and thus provide a real-time view into all the different components present in the environment and drift associated with them. A dynamic SBOM is a key first step to discovering your Log4j related vulnerabilities. You can then search the dynamic SBOM for Log4j packages and instantly find out all of the Log4j instances in your environment.

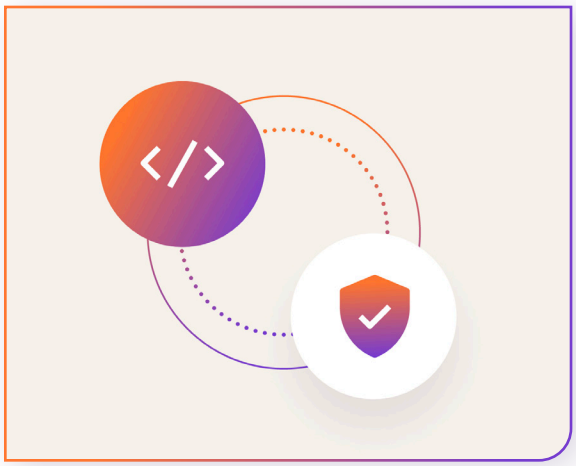


STEP 2: **Establish Exploitability Context**
Just because you discover Log4j packages within your environment does not mean that your environment is vulnerable. The instances have to be exploitable. The dynamic SBOM along with the VEX establishes the exploitability context informing what is exploitable and what is not exploitable. Further mapping back to the originating source on each file system, allows establishing linkages between memory and files, packages, libraries, JARs (nested or otherwise), etc. This association is quite powerful as it establishes the context necessary to determine exploitability of the Log4j instance.



STEP 3: **Remediate**

Now that you know what instances of Log4j are exploitable in your environment, it is time to take action against them. This approach to remediation reduces patching efforts and remediation timelines from months to days, and allows them to quickly focus on critical vulnerabilities that pose a threat to their environment.



For more information, visit www.rezilion.com/platform/dynamic-sbom/ and see it in action and book a demo at www.rezilion.com/request-a-demo/.

About Rezilion

Rezilion's platform automatically secures the software you deliver to customers. Rezilion's continuous runtime analysis detects vulnerable software components on any layer of the software stack and determines their exploitability, filtering out up to 95% of identified vulnerabilities. Rezilion then automatically mitigates exploitable vulnerabilities across the SDLC, reducing vulnerability backlogs and remediation timelines from months to hours, while giving DevOps teams time back to build.

Learn more about Rezilion's software attack surface management platform at www.rezilion.com and get your 30-day free trial.