Snyk Container
A look at Linux package security in containers

Vulnerabilities are tricky things. Vulnerabilities exist in software and hardware, and even just focusing on software vulnerabilities, discovering and handling a vulnerability varies a great deal depending on whether you have a vulnerability in your own code, third party code, or packaged software. And containers add a new wrinkle to this problem because they typically have some packaged software, things like operating system system packages and development tools; plus, your own code; and third party code like frameworks and language packages your apps rely upon.

In this paper, we want to go deeper into the types of Linux package vulnerabilities that occur frequently in containers, because they are a class unto themselves. For a security team like Snyk's, there is extra work required to properly detect and report on Linux vulnerabilities. For example, a general Linux vulnerability that gets reported in the National Vulnerability Database (NVD) can result in differing classifications, severity ratings, and detection and fix methodologies, based on how each Linux distribution chooses to handle fixes. Then there are different methods of exploiting Linux vulnerabilities that are not always obvious by just looking at the NVD reports. This is particularly true for the typical container user, who by and large is not an operating system maintenance expert. These details are important in their work, but if they are too cumbersome to uncover they are less likely to get fixed in the container images.

Container vulnerabilities: What does good look like?

Most commercial and well-established Linux distributions (distros) - Debian, Ubuntu, and Red Hat Enterprise Linux, for example - have dedicated security teams. These teams are responsible for checking CVEs (Common Vulnerabilities and Exposures), prioritizing and implementing security fixes, and maintaining the general security health of the distro. In practical terms what this means is that a vulnerability might exist in a common Linux package, *bash* for example, but each distro could have their own way of addressing this vulnerability or using bash (maybe even leaving bash out altogether). So one cannot simply assume that a vulnerability in bash affects every Linux distro the same way. Not only can the result differ from one distro to the next, but within a distro there are typically a number of supported versions available at any given time. Each version may handle the affected package in a slightly different manner or they could be running different versions of a package.

To properly detect and provide reliable fix information for a these kinds of vulnerabilities, security tools need to take all these differing data sources into account. This means security research teams like Snyk's require a deep understanding of the different internal processes

of each of the established Linux distributions in order to reliably detect the presence of a vulnerability in a container and help users understand how to fix it.

For example, CVE-2019-18276 is a vulnerability affecting the *bash* shell in Linux and is rated a High severity issue in the NVD; in fact, there are exploits readily available. However, if you look at Debian or Ubuntu's analysis of this issue, they both rate it as Low severity; Debian has some releases where the issue is fixed and others that are vulnerable but unlikely to be fixed and Ubuntu has deferred fixes for all releases. All this means that in order to properly report on this vulnerability Snyk needs to do the following:

- Determine what distro and version you are using
- Analyze which packages you have installed in the image - in this case the specific bash version
- Ensure that's the *actual* version of bash in the final image - after all - you might manually install a newer version of or install some other package that overwrites the original bash version
- Report the severity based on the distro's specific guidance (or NVD if there's no data from the distro)
- And, finally, provide relevant fix information, which for Snyk means not just the version of *bash* you might install but also providing information about whether the vulnerability is in the base image (and if so, providing potential upgrades) or if it's in a layer the user added, where we can potentially provide the relevant Dockerfile command that introduced the vulnerability

We can look at Linux vulnerability data in container security solutions through the lens of key security database evaluation characteristics:

- Completeness
- Timeliness
- Accuracy, and
- Actionability

Completeness

When assessing the completeness of vulnerabilities in software that is packaged for a specific Linux distribution, understanding what vulnerabilities are included is crucial. For example, some distro maintainers' security feeds only include triaged vulnerabilities that already have a fix or a patch. On its own, this would only provide a partial view into vulnerabilities affecting those distros. It's important to have *all* vulnerabilities affecting the distro, even without a CVE, whether or not they have received a fix.

In some cases it's also important to go beyond the Linux distribution's security advisory, if it's known to be lacking, and add additional vulnerabilities that affect that distribution.

Alpine Linux is a good example of that, where only a handful of important packages are monitored and reported on by Alpine's security team.

In addition, software may be installed in containers manually, by directly downloading or copying files instead of installing it through the package manager or repositories of the distro. To be able to find vulnerabilities in the manually installed software, the container security solution's security data must also include the upstream advisories for the software being installed and not just those provided by the security team of the distro. In Snyk this is called "key binaries".

Timeliness

Vulnerabilities in Linux distributions go through a process: typically when they are first discovered they are disclosed privately to the maintainers; then publicly shared via NVD or other public vulnerability databases; then the various distro maintainers have a security advisory team that triages the report for their supported release; and then the distro maintainers have their own processes to assign a relative priority level in the distro's context. Security solution providers like Snyk can use this distro-specific priority level to properly report on the vulnerability status and the fix.

Because this process is neither a single step, nor is it strictly linear, timeliness isn't just about adding a vulnerability as fast as possible, but also quickly publishing updated data on existing vulnerabilities and the differences in severity amongst distros.

Timeliness also includes the ability to cover new distro releases quickly. As noted above, each distro maintains their own vulnerability data, but it's actually a bit more complex than that: each distro maintains several feeds covering all of their supported releases at any given time. So that one "Linux" vulnerability is not just a single vulnerability for each distro, but can, in fact, be one vulnerability *per release* of each distro. Again, each release might have its own severity and fix information.

Accuracy

Identifying the existence of a vulnerability, and not flagging packages that are not truly vulnerable, is critical. Put another way: false negatives and false positives are both bad and need to be avoided. In containers this can be particularly tricky for several reasons:

- Containers are really just a layered filesystem: because of this, it's possible for a vulnerable file to be present in one layer, but not actually available in the final image. If we were to flag this file, it would likely result in wasted effort to resolve a vulnerability that would never run in the container image. If this was a secret it would be a different story: while the secret would not be accessible in the running container, anybody who has access to the image could find the secret. They're two different security risks that require different types of analysis.

- There are multiple types of vulnerabilities you can have in a container: operating system packages are one and the main focus of this paper, but you could have files that are copied in, and your own code and dependencies could have vulnerabilities as well.
- You cannot simply assume a vulnerability exists in a container based on its distro. If your scanner detects that you have an image based on Debian 9 ("Stretch") it would create many false positives if the scanner were to flag all Debian 9 vulnerabilities against that image.

Actionability

It is important that the output of all the security research and detections above is readable and accessible for the intended users, containing all possible information about the vulnerability, but also providing the most relevant information up front. From a security perspective, the severity rating is the most obvious (using the distro's guidance, of course) but also making items like fixability and exploit availability front and center. With containers, where you might start with 100s of vulnerabilities, getting to the ones with the most risk is a critical first step. Fixability in containers is also complicated by the fact that, while a fix may exist, in a container you may inherit vulnerabilities from base images. So you (hopefully) will not start installing patches in your containers, but instead look for a newer build of the base image or a slimmer base image with fewer vulnerabilities.

As you assess a container security solution and its security dataset, be sure both the NVD and individual distro metadata are included, specifically looking into the NVD severity and the urgency or impact that may be assigned by some distros. Also look for important details you want to use to focus users' attention on: exploit maturity, fixability, etc. And then make sure that data is usable to those users, without requiring them to become Linux security and maintenance experts on top of everything else they do.

Snyk methodology for Linux vulnerabilities in containers

The Snyk Security Research Team applies a multistep process to generate a measure of normalization across distinctly different distro approaches. This is done to provide a single dimension view to better aid understanding and prioritization of identified vulnerabilities.

The Snyk Security Research Team process is as follows:

1. Research each of our covered distros, including direct communications with distro security teams, and understand the correct approach to each.
2. Collect all the information from the distros in a complete and timely manner, for all their supported releases.

3. Add additional vulnerabilities where relevant and necessary, in a manner that's consistent and without introducing duplication.
4. Add a priority score, taking into account exploit maturity (maturity of any exploit code, looking at if an exploit has been published in the wild, and does the exploit make the vulnerability more critical or dangerous), references, NVD severity, distro relative importance (see above for further details), curated titles, etc.
5. Present all vulnerability information in a way that's easy to consume. For example, if you use Ubuntu, their "priority score" is used as the severity rating for a vulnerability if it differs from the NVD, but the NVD score is shown in the notes as well, for complete visibility.
6. Add additional key binaries (see above for further details).

Much like raw data requires a mix of processing and nurturing to become readily available and useful information, developing a clear lens to a collection of differing Linux distributions and releases requires expertise, research, and curation. Through an extensive base of security experience and continuously tuned processes, the Snyk provides the Linux and container security prowess required to make it possible for developers and security teams to use container, and stay secure together.