7.6 Assignment: Open Source Vulnerability Lab

* A brief summary of the lab activity

In this lab exercise we performed an Open Source Composition Analysis and learned how to use the OWASP Dependency Check tool. We opened an account with GitHub to document our work, communicate with other developers for advice and test the code for the CVE that needed updating. We executed some apache code libraries for the project and learned that if not patched in a timely manner, these libraries can break our applications. Also, system vulnerabilities checks can not detect vulnerable code libraries because traditional vulnerability scans check for vulnerability of installed applications and exposed network services. We checked the JSTL XML tag for CVE 2015-0254 Which allows attackers to execute arbitrary code or conduct external XML entity attack.

We then created a Repository in GitHub to work on the project and update the fork to correct out-of-date code to most-up-to-date code. Then we created a Dependency-check-plugin.html with code from NVD.

* Analysis:
  + What are some strategic limitations of the demonstrated tool with regards to determining overall risk exposure?  How would you overcome those limitations?

An important consideration in using Dependency Check tools is to monitor 3rd party libraries and keeping them up-to-date. Most times the libraries author has been notifies and patched the vulnerability, but developers need to also practice due diligence and check updated bug reports and release notes. OWASP Top 10 is a globally recognized solution for developers as a reliable resource. It names Configuration Vulnerabilities and Using Components with Known Vulnerabilities as a common cause of exploitation by hackers in accessing systems and data. (Van der stock et al.)

In misconfigurations, default configurations, incomplete ad hoc configurations and misconfigured HTTP headers exposes sensitive data. Cross-site scripting XSS can occur when untrusted data is injected to a web page without proper validation or updates an existing webpage using a browser API that can create HTML or JavaScript. Attackers can then execute scripts in the victim browser which can hijack user sessions, deface wed sites, or redirect traffic to malicious sites. (Van der stock et al.)

* + Assuming that South Balance is using one or more of the vulnerable open source libraries, how would you determine the range of potential impacts for South Balance?

Using the OWASP Dependency Check Tool is the first step and learning how to interpret the dependency-check-report.html. The Dependency Check tool searches through the project’s packaged libraries and matches them to CPEs to validate that the CPE is valid. After it is validated then it identifies any CVEs that are known matches in NVD. This report will help in determining if any vulnerabilities exist and evaluate if they are a threat to my SB project. (Van der stock et al.)

If it is determined that the reported vulnerability is not harmful to the SB project, then it can be recorded as an exception in a suppression configuration file and the Dependency tool will no longer report them. By having the Dependency Check tool integrated into SB CI/CD communications it can help identify vulnerabilities as soon as they become available to the public in the NVD. (Sherwood)

The use of Dependency Check provides valuable time to security teams by automatically identifying potential threats in libraries and provide links to aid in the remediation of any actual vulnerabilities as they are identified and made public to the NVD. (Sherwood)

* + How would you mitigate risks if South Balance is using one or more of the vulnerable open source libraries but they could not be upgraded without breaking the Web Application?

Most times, if developers are still using older versions of a library, upgrading might mean switching to a newer version. Since they are typically not backwards compatible, the newer version introduces more risks and requires more dev effort. (Podjarny)

Since most dependencies are indirect, the best way to perform an indirect upgrade is through a direct one. If your tooling doesn’t support direct upgrading then you’ll have search manually to find a fix. Another potential obstacle to upgrading is a conflict. Upgrading one library might trigger a conflict with another. There are global dependency managers that allow the parent app to add a constraint. Constraint are a safe solution as long as your ecosystem tooling can figure out a conflict-free combination of libraries. (Podjarny)

Patching is the next best solution after upgrading. The best place to look for a patch is the new version of the vulnerability package. Another common source for patches are external pull requests (PRs). Then forking the GitHub repository to a user you control and patching it.

If you can’t fix the vulnerability code, you can attempt to block attacks that attempt to exploit it instead by introducing a rule in the web app firewall by modifying parts of your app that accepts related user input, or blocking a port. (Podjarny)

REFERENCES:

Podjarny, G. (2018). Mitigating known Security Risks in Open Source Libraries. Fixing vulnerable open source packages. OReilly. 2018, January 30. Retrieved from: <https://www.oreilly.com/content/mitigating-known-security-risks-in-open-source-libraries/>

Sherwood, S. (2017). Discover Vulnerabilities with Dependency Check. UnderTree.io. 2017, February 9. Retrieved from: <http://undertree.io/security/2017/02/09/monitoring-vulnerabilities-with-dependency-check.html>

Van der Stock, A. Glas, B. Smithline, N. Gigler, T. (2017). OWASP Top 10 | Top 10 Web Application Security Risks. OWASP. <https://owasp.org/www-project-top-ten/>