Documentation and Tracking of Vulnerabilities in Open-Source Libraries

Nicholas Chan and John A. Chandy
University of Connecticut

Position Paper Area: *Guidelines outlining security measures*

A clear and observable trend within the software industry is the increasing use of open-source software components. The use of open-source components within a software product confers multiple benefits. In addition to the added convenience of using pre-written functions and objects, the transparency of being open-source allows for increased scrutiny of the source code from the developer community resulting theoretically in code that is more robust, reliable and secure. As such, open-source components have been adopted by multiple industries ranging from software and communication companies to the defense and public/government sector. However, like proprietary software, there still exists vulnerabilities within open-source software. Even with the increased scrutiny open-source software undergoes, coding mistakes are bound to slip through. Some of these mistakes can be leveraged by an attacker to compromise the security of the software and cause it to perform in unintended ways that would benefit the attacker. For instance, the most common type of vulnerability are buffer overflow vulnerabilities where an attacker writes data to a buffer larger than the size of buffer overwriting the call stack including the return pointer allowing the attacker to execute malicious code. When these open-source libraries are used to create software that is going to be used by both government and private organizations, steps need to be taken to ensure that these open-source components will not introduce significant risk.

Vulnerabilities have a life cycle that can be broken down into several distinct phases. The beginning of this cycle is the discovery of the vulnerability whether it is the vendor, a third party, or hackers. During the disclosure phase, the details of the vulnerability are published and made known to the public. The phase that follows is critical as the vulnerability is now publicly known and there exists no current fix. In that time frame, code that exploits these vulnerabilities is sometimes available in places such as Metasploit and ExploitDB. Around 4 percent of these vulnerabilities have the exploit code made available within a year after disclosure and of those exploits, 75 percent of them are made available within 28 days. The patching phase begins once the vendor of the affected software releases a patch that fixes the vulnerability. The life cycle of a vulnerability only ends when the patch is universally adopted by all users of the software.

Many of these vulnerabilities are documented in the Common Vulnerabilities and Exposures (CVE) database systems maintained by MITRE and NIST. These CVE entries are composed of unique IDs, descriptions of the vulnerabilities, as well as scores that rate how serious these vulnerabilities are. These entries also detail the versions of the software product that are affected. It is also common for these CVE entries to detail how to remediate the vulnerability. For instance, a CVE entry might detail the version of the software that implements a fix for the vulnerability. While the NIST database provides much useful information, there are shortcomings when it comes to how well the vulnerabilities of third-party libraries are documented. As part of a study, we examined the top 100 C language libraries found on GitHub looking for patterns in CVE documentation.

Out of the 100 libraries examined, CVEs were only found for 15 of them. Given that bugs are a regular occurrence when it comes to any piece of software, it is highly unlikely that the remaining 85 libraries contain no security vulnerabilities of any kind. Given that these are the top 100 most popular libraries, there should be sufficient scrutiny of the source code such that any existing vulnerabilities can be detected before any malicious attacker can exploit them. This observation should be investigated further to examine why so few of these libraries have documented CVEs.

From the libraries with documented vulnerabilities, 447 associated CVEs were found. From this dataset, only 248 of them detailed a fixed version of the library. Moreover, from that set of CVEs, only 130 mentioned the associated CVE in the GitHub commits that implemented the fix. This key observation highlights issues when it comes to the documentation of CVEs. Of the 15 libraries that were examined, only 3 of them specifically mentioned CVEs in the GitHub commit logs. Even within these 3 libraries, not all the CVEs were found in the commit logs highlighting inconsistent practices. For the purpose of informing the users of these open-source libraries of security vulnerabilities, the CVEs should be documented within the GitHub commits.

When a vulnerability is disclosed, there is a time gap between when it is disclosed and when its remedy is widely adopted. The median time between disclosure and the publication of metadata such as affected software and versions is 35 days. To ensure that a vulnerability reaches the end of its life cycle as fast as possible, the relevant patches must be rapidly and universally adopted. Part of that process involves informing the end users of the vulnerabilities and the risk they might pose. In analyzing social media discourse of security vulnerabilities, Schiappa found that much of that discussion begins on GitHub before reaching other platforms and found a positive association between GitHub events and exploit related articles. The results of Schiappa's research highlights the importance of GitHub when it comes to informing end users of software vulnerabilities. GitHub commit logs are another potential avenue to expand this form of communication.

We strongly recommend that the open-source community develop a standard for how security issues and fixes are communicated to the end users. In addition to informing users, more consistent documentation of CVEs in GitHub commits makes easier the implementation of automated tools that analyze the risk of vulnerabilities in source code. For software projects that are large in scale and involve heavy use of open-source libraries, the process of determining the risk of the CVEs from these libraries needs to be automated. There exist some automated tools that serve this purpose such as the OWASP Dependency Checker and Vulas. These software compositional analysis tools examine the libraries a project uses and check if the Common Platform Enumeration (CPE) matches the CPE of any CVEs from the NIST database. The effectiveness of these tools is contingent on consistent logging standards for CVEs such that a program can easily identify matching CVEs for a given library. Furthermore, accurate tracking of vulnerabilities tied to library version numbers can allow end users to examine software "bill of materials" to determine if their software packages contain out-of-date libraries that contain known vulnerabilities.