Evaluating Security of Software Through Vulnerability Metrics

Jonathan Sönnerup, Martin Hell
Dept. of Electrical and Information Technology
Lund University
P.O. Box 118, 221 00 Lund, Sweden
Email: {jonathan.sonnerup, martin.hell}@eit.lth.se

Abstract—Understanding and measuring security of software in terms of vulnerability metrics is important when reviewing and deciding between softwares. The large number of disclosed vulnerabilities will continue to expose software intensive systems and products to attacks, and the choice of third party software will affect stability and reliability of products incorporating this software. We collect CVE data from NVD and version release data from GitHub in order to study how vulnerabilities, exploits and patches affect the exposure of software. By combining all data for each software we propose a software vulnerability exposure score that can be used when evaluating security. We perform a large-scale study of more than 37000 software and also analyze common web servers and cryptographic libraries in more detail. We show that the proposed score is both diverse and close to normally distributed, making it attractive as a review and comparison tool.

Index terms – Security exposure, exploit, vulnerability life-cycle, patch, NVD

I. INTRODUCTION

Software is a key component when designing new products. In a survey [18], it is shown that 50% of products’ code base originates from open source software (OSS). It is also shown that 50 percent of companies do not have a formal policy for selecting and approving open source code, and that more than 30% of companies have no defined process for identifying, tracking or remediating known open source vulnerabilities. Since the reliability of delivered products is based on the reliability of its components, it is important for the open source software to be reliable as well.

Several variables must be taken into account when selecting software, such as licenses, its age, how well it is maintained, the number of contributors, and the quality of the documentation. Another variable to measure when choosing software is the level of security, where a high level of security is assumed to be more stable and reliable compared to software with low level of security. The focus of this paper is how to measure software based on vulnerabilities, exploits and how vulnerabilities are mitigated.

There are a number of publicly available resources for information related to vulnerabilities, e.g., [2, 11, 13, 15]. This information is not centralized, and in order to get a complete picture different sources need to be consulted. Moreover, while some information can be automatically retrieved through generic means, other need more tailored approaches.

In this paper, we develop and evaluate methods for retrieving data related to some aspects of software vulnerability management. The metrics that we use are vulnerability severity, vulnerability frequency, the existence of exploits and the exposure time. We divide the data into two complexity levels based on how easy it is to retrieve the information. We carry out a large-scale analysis of 37116 softwares and propose a simple scoring for determining the security of the software with respect to the targeted metrics. This scoring is configurable, diverse and approximately follows a normal distribution. We further analyze a set of common web servers and cryptographic libraries using our proposed score. The outline of the paper is as follows. In Section II we give the motivation and goals of the paper and in Section III we describe some related work. Then, in Section IV we start by detailing the security metrics that we focus on in this paper, before we further discuss the methodology for retrieving the information for the metrics in Section V. The results and their interpretation are given in Section VI and the paper is concluded in Section VII.

II. MOTIVATION AND GOALS

The motivation for this work is based on three observations. First, software vulnerability information is not centralized. The most important and widely used source of information is the National Vulnerability Database (NVD), which collects information on vulnerabilities. Each vulnerability is associated with a CVE identifier, a CVSS score, and other related information. Exploit information can be found in a variety of sources, most notably Exploit-DB [2] and SecurityFocus [15]. Further, software repositories such as GitHub can be used to find information regarding the actual software, while security advisories often require an analyst to consult the vendor, or another third party, website for the software. The Open Source Vulnerability Database (OSVDB) has previously been used to collect more complete information on vulnerabilities, such as exploit dates and patch dates [16, 4], but this database is not publicly available anymore. The lack of centralized infor-
formation makes it much more difficult to assess vulnerability information for softwares.

The second observation is that sources such as NVD and Exploit-DB only provide information on single vulnerabilities. When evaluating software in terms of security, single vulnerabilities do not provide enough information. The overall number of vulnerabilities and the rate at which they are found can be used to assess the overall code quality and security awareness among the developers.

Third, the tools used for retrieving information must sometimes be tailored to specific software or specific vulnerabilities to different extent. The CVSS base score and vulnerable products can be easily retrieved for all software through data feeds or APIs, while the time between a CVE publish date and the release of the patched version will require an approach more tailored to the specific software.

Based on these observations, the goals of this paper are as follows.

- Investigate to which extent certain data can be retrieved using one generic program through well defined data feeds or APIs, and how much software or vulnerability specific modification is required in order to find more, or more accurate, data.
- Summarize software metrics for a large set of softwares based on the defined vulnerability data and define a score for software vulnerability exposure.

III. RELATED WORK

The most well known and widely used metric for measuring the severity of a vulnerability is the CVSS. This scoring system was proposed in [1] and later improved as CVSS version 2 in [10, 14]. The most current version is CVSS version 3 [3]. The goal of CVSS is to have an open, transparent metric, which is objective in the sense that if two analysts review a vulnerability, they will score it equally (reproducibility). An important aspect of a metric is that it is diverse, since this will increase the amount of information it provides. Attempting to improve the diversity of CVSS, a different weighting of the involved components was used in WIVSS [19]. We use the well established CVSS score as is, but consider the diversity to be an important aspect of a severity score.

Another scoring system, VRSS, is presented in [8]. It uses a mixture between two qualitative scoring systems (IBM ISS X-Force and Vupen Security) and the quantitative CVSS. Based on experience, the authors argue that vulnerability scores should be normally distributed, and the qualitative systems are more consistent with this. However, the quantitative systems are not transparent. The analysis shows that VRSS conforms to the normal distribution.

Using individual software vulnerabilities, and e.g., their CVSS scores, there have been proposals for measuring the security of a computer system, or a product consisting of a set of software modules [5, 6, 17]. The relationship between the number of vulnerabilities and a system’s attack surface is shown in [9].

Instead of aggregating vulnerabilities on a system level, another approach is to aggregate this information, and possibly more information, for individual softwares. This is similar to the approach that we take in this paper. Previously, one such metric was proposed by Wang et al. in [20]. The metric uses weaknesses described in CWE and gathers the weaknesses that cause most vulnerabilities. The authors denote this the “representative weaknesses”. The final score is based on the number of vulnerabilities, the average CVSS score, the percentage of each weakness, and the timespan from the first to the last vulnerability. Our approach is to consider all vulnerabilities for a software, not restricting us to CWEs with most vulnerabilities.

Large-scale analysis of vulnerabilities has been considered in [4]. The authors look at several different metrics related to vulnerabilities, e.g., exposure time, date between discovery and disclosure and date between disclosure and exploit. Several sources of information is used in the paper which are not available today, e.g., OSVDB which is no longer openly accessible. Moreover, vulnerabilities are only considered in isolation, while we map them to software names through the CPE listings. Another large-scale analysis was performed in [16], also using OSVDB in addition to NVD and the data collected in [4]. They categorize vulnerabilities according to e.g., weakness types (CWE) and software categories, but do not attempt to look at metrics for software that depend on the disclosed vulnerability data.

IV. SECURITY RELATED INFORMATION

Measuring security of a software requires measurement of security-related information. Here we focus on information related to vulnerabilities, exploits and patches. We stress that this is only one important aspects when reviewing the suitability of third-party software in devices or products. In particular we study four different metrics.

Vulnerability severity. This measures how critical a vulnerability is. It will depend on several factors, such as its impact in terms of confidentiality, integrity and availability, if exploited. Also access complexity and prerequisites for launching an attack, as well as to which degree an attacker needs to authenticate to the system before launching an attack is included in the score. The CVSS score is the most well-known and widely used metric for the severity of a vulnerability.

Vulnerability frequency. This measures the rate at which vulnerabilities are disclosed. The underlying reason for a high vulnerability rate could be the popularity of the software, its complexity or a lack of security awareness among developers. Other factors incentivising attackers or researchers to find new vulnerabilities, such as bug bounty programs, could be a source of frequent vulnerability disclosures. In any case, software with a very high frequency of vulnerabilities are in general more exposed to attacks.

Existence of exploits. An exploit is a piece of software that takes advantage of a vulnerability. Not all vulnerabilities have publicly available exploits. This could e.g., be because it is difficult to exploit the vulnerability or that the impact is not
high enough for it to be interesting to develop exploit code. While a lack of publicly available exploits do not exclude the existence of exploits, it does make the vulnerability harder to exploit for the general audience.

**Exposure time.** The typical remediation when a new vulnerability is found is for the vendor to patch and release a new version of the software. Under the responsible disclosure model (see e.g., [7] for guidelines on how vendors should best enable the model), the vendor is notified of the vulnerability before it is made public. This will give the vendor an opportunity to develop and release patches to the software before the vulnerability information is widespread. However, sometimes patches or new releases are not available at the time of publication. This could be due to the fact that responsible disclosure was not applied, that the vendor did not respond properly to the bug report or that the time window was not large enough for a patch to be developed in time. We define the exposure time as the time between public announcement of the vulnerability and the release of a patch. While the exposure time can be very situation dependent (and out of the vendor’s control), large exposure times will indicate that the software is in general less secure. In many cases the user do not update immediately, or at all. This aspect is not taken into account here, since we are considering security from the software development team’s point of view, not the user’s.

V. METHODS OF DATA RETRIEVAL AND COMPLEXITY LEVELS

We introduce two complexity levels for the information retrieved.

- Data retrieval at complexity level 1 includes data that can be easily accessible through a well-defined API, or similar. It must also be possible to use the same generic program to gather data for all software. Such data retrieval allows for software analysis at large scale.

- Data retrieval at complexity level 2 includes methods that require more tailored or complex approaches, e.g., scraping information directly from websites, or modifications of the analysis tool to make it compatible with different software or products.

In order to measure and compare the security between different software, we need to define a combined security metric. Which aspect is most important will depend on the use case, and different reviewers may have different preferences. In contrast to previous work [20], our main goal is not to propose a completely defined scoring system. Instead, we allow the contrast to previous work [20], our main goal is not to propose a completely defined scoring system. Instead, we allow the

The individual metrics can be weighted arbitrarily depending on their importance, but for simplicity we will use \( a_i = 1/N, \forall i \) in this paper. Moreover, we use four metrics \( x_1, \ldots, x_4 \), corresponding to the features discussed in Section IV, but the score can trivially be extended to using more metrics based on other features. It is also possible to limit the analysis to only look at vulnerabilities from the last few years. This will better capture the current situation for the software. To capture as much information as possible, we would like to have each individual metric as diverse as possible. At the same time, following [8], it is desirable to have a score following a normal distribution and thus have the individual features to have this property as well.

Data is retrieved through the NVD database and GitHub. NVD has accessible data feeds in both XML and JSON format, and GitHub provides an API for information about repositories. This makes it easy to retrieve information from both sources. However, as will be seen, this will not be fully adequate for the information that we seek.

A. Vulnerability Severity

The NVD database is a well-known and often used source of vulnerability information. Each CVE is individually analyzed and given a CVSS score. The score is quantitative in nature, based on well defined sub-metrics, and is designed to be reproducible. In addition to the CVSS score, each CVE also includes a list of vulnerable software and which versions are affected. This is given using the Common Platform Enumeration (CPE) naming scheme. The CPEs are used to find the CVSS for a particular software.

The CVSS score is given for each individual vulnerability, so in order to get a metric for the severity of the complete set of vulnerabilities for a software, we take the average CVSS score across all \( M \) vulnerabilities,

\[
x_1 = \frac{1}{M} \sum_{i=1}^{M} \text{CVSS}_i.
\]

This information can be found in the data feeds from NVD and its retrieval is similar for all software and CVEs. It can thus be seen as complexity level 1.

B. Vulnerability Frequency

Software with many vulnerabilities are more exposed and may also indicate a lack of security awareness in the development process. Still, software that has been around for many years are more likely to have more vulnerabilities than newer software. Thus, we measure the number of vulnerabilities per time unit. The starting date used is defined to be the date of the first vulnerability for that particular software, while the ending date is the current date. The time unit is given in months and thus the metric \( x_2 \) used will be the average number of vulnerabilities per month since the disclosure of the first vulnerability. This number will be scaled to fit into the interval \([0, 10]\). Similar to vulnerability severity, this information is retrievable from the NVD data feeds and thus belongs to complexity level 1.
C. Existence of Exploits

The existence of publicly available exploits will make it easier to mount attacks. The metric \( x_3 \) used here is the fraction of vulnerabilities that also has a publicly available exploit. The presence of such exploits also indicates that the vulnerabilities found for the product are easily exploitable, and since the presence of exploit code is not captured by the base CVSS score, this metric adds information regarding the security of the software. The NVD database includes a set of relevant links for each CVE and if there are known exploits, links to these can be found here. To each link, a reference type is sometimes also added, describing what the URL links to. One such reference type is exploit and other examples include vendor advisory and third party advisory. However, the reference types are not present in the data feeds. In order to find the type, the website needs to be scraped. Thus, we scrape all CVE web pages and extract the links together with their type. Due to the large number of CVEs, this is time consuming and very inefficient compared to retrieving data directly from the JSON or XML feeds. Thus we regard this as complexity level 2. Still, gathering statistics for which URLs are most often used to point to exploit code, it is possible to only use the data feeds, looking at the URLs. This will result in data retrieval using complexity level 1.

D. Exposure Time

The exposure time will measure for how long time a vulnerability has been public without existing remediation. Three different data points are used to measure this time. First, we consider the published date of the CVE to be the date of publicly announcing the vulnerability. Second, we use the CPE list for the CVE to retrieve the most recent version of the software to be affected by the vulnerability. Third, the GitHub API is used to retrieve the release dates for the software releases. 5000 API requests per hour are allowed for the GitHub API is used to retrieve the release dates for the software releases. The exposure time is then given by the time between the CVE release date and the release date for the software release following the most recent vulnerable release. The metric \( x_4 \) is given by the average exposure time for all CVEs for that software, scaled to the interval \([0, 10]\). One limitation of this approach is that GitHub software rarely use release or tag names that correspond to the name in the CPE string. Thus, for each software, the program retrieving the data must be adjusted in order to map between the formats. This limits the possibilities to perform a large-scale comparison. Thus, we consider this as corresponding to complexity level 2. However, when comparing and reviewing a small set of individual softwares, such adaptations are feasible.

E. Reviewed Software

In total, 39822 unique software names are identified from NVD. Our collected metrics will naturally depend on the size of the software, and in order to remove some outliers, we filter out operating systems and firmware. After filtering, 37116 softwares remain, with a total of 96321 vulnerabilities.

<table>
<thead>
<tr>
<th>Software</th>
<th># CVE</th>
<th># Exp.</th>
<th>Software</th>
<th># CVE</th>
<th># Exp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Htp Server</td>
<td>214</td>
<td>51</td>
<td>MatrixSSL</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>Nginx</td>
<td>14</td>
<td>5</td>
<td>mbed TLS</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Lighttpd</td>
<td>28</td>
<td>9</td>
<td>PolarSSL</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Apache Tomcat</td>
<td>163</td>
<td>35</td>
<td>OpenSSL</td>
<td>183</td>
<td>16</td>
</tr>
<tr>
<td>Nodejs</td>
<td>28</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For complexity level 1, we use the data feeds from NVD. This allows us to compute metrics for the complete ensemble of software that has any vulnerability recorded in a CVE. It also allows us to analyze the distribution of the metric, verifying if it is diverse and normally distributed.

For complexity level 2, we additionally use the metrics for exploits and exposure time retrieved using GitHub, and compute metrics for a chosen set of software, namely some popular web servers and cryptographic libraries. According to a survey by Netcraft [12], there are now (2018) 1,770,411,187 available websites, with around half of them using transport layer encryption [21]. These numbers motivate the ongoing need to compare the quality of these products. The software used in the analysis are summarized in Table I.

VI. RESULTS

As a starting point, we look at the distribution for the number of CVEs the different software have. This distribution is shown in Fig. 1a. It is clear that the vast majority of software have only one or very few vulnerabilities. Note that there are no products for which there exists zero vulnerabilities, since we are only analyzing products with vulnerabilities. Upon closer inspection, the majority of the software with only one vulnerability are small software and software with a very small number of installations. The outliers in Fig. 1a, i.e., the softwares with a CVE count above 30 are typically well known and widely used software such as Adobe Flash Player (852 CVEs) and Chrome (1369 CVEs).

A. Metrics for Complexity Level 1

We start by analyzing the results for the metrics on complexity level 1, i.e., vulnerability severity and vulnerability frequency.

A histogram showing the average CVSS per software, for all 37116 identified software modules in the set, is given in Fig. 1b. Since the CVSS score already is a score in the range \([0, 10]\), there is no need to scale the results to this range. We can see that the distribution is somewhat similar to the total distribution of individual CVSS scores, with values in ranges 4-5, 7-8 and 10 being higher than expected from a normal distribution. The mean value is \( \mu = 6.35 \) and the standard deviation is \( \sigma = 1.76 \).

\(^1\)http://www.cvedetails.com
Looking at the distribution for the number of vulnerabilities per software per month, as given in Fig. 1c, shows a distribution that is close to exponential. This is due to the fact that there are many softwares that have only one or very few vulnerabilities. In order to convert this feature as a metric that can be used in Eq. 1, we need to transform it such that the metric is in \([0, 10]\). To also make it diverse, we compute the metric as

\[
x_2 = 10 \cdot \frac{2.20}{2.20} \cdot \ln(\ln(x) + 6.49).
\]

Taking the logarithm of the number of vulnerabilities per software per month for all software yields a set of values where the smallest value is found to be -5.49. In order for the outer logarithm to be defined and non-negative, we add the offset 6.49, i.e.,

\[
\ln(x) + 6.49 \geq 1.
\]

The largest number in the set is 2.20. Thus, the coefficient \(\frac{10}{2.20}\) scales the score such that

\[
0 \leq f(x) \leq 10,
\]

where 2.20 is given by the maximum value of the unscaled function. The coefficients are based on current and historical data. Fluctuations may occur which results in a score lower than 0 or higher than 10. In these cases, the results are to be capped both in the lower end and in the higher end. A histogram of the transformed scoring is shown in Fig. 1d. The transformation does not change the order of modules with respect to the score, due to the logarithm function being monotonically increasing.

Weighing the two features equally (i.e., \(a_1 = a_2 = 0.5\) in Eq. 1), we get a score distribution shown in Fig. 1e. As can be seen, this distribution is very close to normal, but with somewhat higher concentration around 4.5. It also shows a high degree of diversity. This makes our score suitable for looking at these metrics and extracting information regarding software vulnerabilities and their severity over time.

### B. Metrics for Complexity Level 2

To determine if a vulnerability has a public exploit, we make the assumption that an exploit exists if there is a link with reference type marked *Exploit* on NVD. Otherwise, no exploit exists. Of the 96321 vulnerabilities analyzed, 25096 have at least one reference marked as exploit. To view this as a metric, we count the fraction of vulnerabilities with an exploit, per software. The fraction is in the range \([0, 1]\), so in order for the metric to be in the range \([0, 10]\), we multiply the result by 10,

\[
x_3 = 10 \cdot \frac{\# \text{Exploits}}{\# \text{CVEs}}.
\]

The distribution of the fraction of exploits per software is shown in Fig. 1f. Since there are many softwares with only 1 or very few vulnerabilities (see Fig. 1a), the number of softwares with fraction 0 or 1 is very high (22324 and 10077 respectively). These have been excluded from the figure.
The exposure time of a software is measured as the number of days between published date and patched date. If the number of days are negative, i.e., the vulnerability was patched before the disclosure, we count it as zero. There is no practical difference between releasing a patch at the day, or before, the disclosure in the sense that the software is not exposed either way.

Measuring the exposure time, for the web servers and cryptographic libraries, yields the results shown in Table II. As shown in the table there are a couple of outliers, both software with no exposure time at all, but also software with very high exposure time. Analysis shows that for all software, there are at least one vulnerability that was disclosed at the release time of patch, or before. This explains why the minimum exposure time is zero for all softwares. As for Apache Tomcat, not all versions are available at GitHub, e.g., only version 7.0.2 for Apache Tomcat 7 is available. This means that we miss several updates and patches, hence the large exposure times. One solution is to scrape Apache’s archives in order to find all updates and dates. Another reason for high exposure times might be due to the fact that GitHub did not exist prior to 2008, hence the exact dates for the software versions might not be accurate. Performing an analysis with a shorter timespan should result in more accurate timestamps for patches.

In order to use the exposure time feature as a metric, we must transform it such that the metric is in $[0,10]$. By following the same principle as in Eq. 3, we take the logarithm of the values. Since the smallest exposure time is 0, we add 1 to prevent the logarithm to be negative. Due to the error in the exposure time for Apache Tomcat, we cap the values at 30. This way, the errors do not propagate too much in the metric. The metric is computed as,

$$x_4 = \frac{10}{3.44} \ln(1 + x).$$  \hspace{1cm} (5)

The largest value after taking the logarithm is found to be 3.44, which is being used to scale the metric to fit the range. By transforming the values for the web servers and cryptographic libraries, the mean value and standard deviation are found to be $\mu = 5.31$ and $\sigma = 4.46$, respectively.

Analyzing the references for each vulnerability reveals that some references are more occurring than others. The top 5 domains are listed in Table III. We further analyze the references and count the number of times one of the top 5 references is included in the reference list, both when there exists an exploit (Exploit) and when there does not (No Exploit). We also count the number of times one of the top 5 is included but no other reference in the top 5 set (Unique), when an exploit exists. We calculate a ratio (Ratio) between the number of times a reference is included when an exploit exists, and the total number of exploits.

As shown in the table, one can not rely only on the existence of a specific reference to get information regarding the existence of exploits. However, analyzing randomly selected vulnerabilities shows that even though NVD has not marked a reference as exploit, references to Exploit-DB are references to exploits and should be marked as such. Therefore, if a reference to Exploit-DB exists, one may assume that an exploit exists. This increases the ratio from 0.26 to 0.36. Thus, one may look only for Exploit-DB references in complexity level 1 to draw the conclusion that an exploit exists. It is not accurate since several exploits will be missed, but it yields some amount of information.

Investigating in which years the references were most seen, we notice that Vupen, Securityreason and Xforce are not referenced by NVD from 2016 and after, whereas both Exploit-DB and SecurityFocus are still active. When analyzing software during a shorter timespan, it is preferable to check only the active references.

Applying the SVE scoring function to our set of web servers and cryptographic libraries, we get the result listed in Table IV. Looking at features $x_1$ and $x_2$, we see that the resulting score is similar for the softwares. The major difference is the frequency at which vulnerabilities are discovered. Taking features $x_3$ and $x_4$ into account yields a more diverse result. This is mainly due to the diversity in the exposure time feature. Since this feature, and its transformation, only depends on analysis from the 9 softwares, it is not accurate and is not applicable for a general case, without being adjusted.

### VII. Conclusions

While certain vulnerability related information can easily be retrieved from data feeds and APIs, the automatic collection of patch dates require more tailored approaches and must be adapted to the specific software. Still, it is possible to use automated tools to retrieve information on vulnerability frequency, severeness, exploits and exposure time. The proposed scoring system based on these metrics is simple, yet both diverse and

---

2https://archive.apache.org/dist/tomcat/tomcat-7/
close to normally distributed. This shows that our approach can be used when measuring and comparing software in terms of vulnerability exposure over time. The score gives an indication of how exposed the software is over time, and how well the development process and the development team responds to vulnerabilities. By tweaking the weighting factors, a software reviewer can easily adjust the score such that it adheres to the reviewer’s preferences and priorities.

REFERENCES


TABLE IV: Metrics and SVE scores for a chosen set of software.

<table>
<thead>
<tr>
<th>Software</th>
<th>avg. CVSS</th>
<th>CVEs / mon.</th>
<th>SVE score 1 $\left( a_1 = \frac{1}{2} \right)$</th>
<th>Exploit % $\left( x_3 \right)$</th>
<th>Exposure time $\left( x_4 \right)$</th>
<th>SVE score 2 $\left( a_1 = \frac{1}{3} \right)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nginx</td>
<td>5.41</td>
<td>6.61</td>
<td>6.01</td>
<td>3.57</td>
<td>0</td>
<td>3.90</td>
</tr>
<tr>
<td>Apache Htp Server</td>
<td>5.42</td>
<td>8.27</td>
<td>6.84</td>
<td>2.46</td>
<td>7.62</td>
<td>5.94</td>
</tr>
<tr>
<td>Lighttpd</td>
<td>5.40</td>
<td>6.89</td>
<td>6.14</td>
<td>3.21</td>
<td>9.23</td>
<td>6.18</td>
</tr>
<tr>
<td>Apache Tomcat</td>
<td>5.11</td>
<td>8.16</td>
<td>6.63</td>
<td>2.15</td>
<td>10.00</td>
<td>6.36</td>
</tr>
<tr>
<td>Nodejs</td>
<td>5.33</td>
<td>8.04</td>
<td>6.68</td>
<td>0.71</td>
<td>3.67</td>
<td>4.44</td>
</tr>
<tr>
<td>MatrixSSL</td>
<td>5.48</td>
<td>8.30</td>
<td>6.89</td>
<td>2.00</td>
<td>0.00</td>
<td>3.95</td>
</tr>
<tr>
<td>mbed TLS</td>
<td>7.15</td>
<td>7.27</td>
<td>7.21</td>
<td>1.25</td>
<td>9.76</td>
<td>6.36</td>
</tr>
<tr>
<td>PolarSSL</td>
<td>5.69</td>
<td>6.76</td>
<td>6.22</td>
<td>0.91</td>
<td>3.67</td>
<td>3.34</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>5.41</td>
<td>8.19</td>
<td>6.8</td>
<td>0.87</td>
<td>7.98</td>
<td>5.61</td>
</tr>
</tbody>
</table>

### Miscellaneous

- **Apache Http Server**: 8.27, 6.84, 2.46, 7.62, 5.94
- **Apache Tomcat**: 8.16, 6.63, 2.15, 10.00, 6.36
- **Nodejs**: 8.04, 6.68, 0.71, 3.67, 4.44
- **MatrixSSL**: 8.30, 6.89, 2.00, 0.00, 3.95
- **mbed TLS**: 7.27, 7.21, 1.25, 9.76, 6.36
- **PolarSSL**: 6.76, 6.22, 0.91, 3.67, 3.34
- **OpenSSL**: 8.19, 6.8, 0.87, 7.98, 5.61

---

**Software**

- Avg. CVSS: Average CVSS score for the software.
- CVEs / mon.: Number of CVEs found per month.
- SVE score 1: SVE score using the formula $\left( a_1 = \frac{1}{2} \right)$.
- Exploit %: Percentage of exploits.
- Exposure time: Time taken to expose the vulnerability.
- SVE score 2: SVE score using the formula $\left( a_1 = \frac{1}{3} \right)$.