Quantifying the Milestones of Cyber Vulnerabilities

Michael R. Lidestri and Neil C. Rowe

U.S. Naval Postgraduate School

michael.lidestri@gmail.com, ncrowe@nps.edu

Abstract: We have developed quantitative models of the durations between the key events in the lifetime of a cyber vulnerability: disclosure time from creation time, path time from disclosure time, exploitation time from disclosure time, and patch time from creation time. Our analysis was based on data from the National Vulnerabilities Database of NIST with additional information from software vendors and the Exploit Database. We observed a significant slowing of patching five years after the release date. We found the cumulative distribution fit well to a Weibull distribution. The duration between disclosure and patching was a symmetric sigmoid curve about zero. We also examined the effects of severity of a vulnerability and operating system under which it occurs. **Keywords**: cybersecurity, vulnerabilities, timelines, distributions, patching.

This paper appeared in the Proceedings of the SAM 22 (Security and Management) conference, Las Vegas, NV, July 2022.

Introduction

Vulnerabilities are software bugs (flaws) that allow a malicious actor to attack the confidentiality, integrity, or availability of an information system. Bugs are frequent in many software programs; an estimated 20 occur for every 1,000 lines of code [5]. Vulnerabilities in software are exploitable for malicious purposes unless they are removed by corrective software updates called patches [20]. Exploits are malicious code that takes advantage of software vulnerabilities to insert a payload onto a target information system. Vulnerabilities that remain unpatched are called zero-day vulnerabilities and the corresponding exploits are zero-day exploits [12].

The opportunity to use or re-use exploits may be limited [24]. Cyber exploits are often considered to have a one-time capability with limited efficacy: After an exploit is deployed and subsequently detected, a patch (fix) will normally be

quickly developed. Once this occurs, the underlying vulnerability is no longer available, and the exploit is ineffective. Therefore, attacks that exploit software vulnerabilities can be described as transitory or perishable [9, 24]. Even when they are not used, exploits may become ineffective if the vulnerability is discovered; this is a trait called obsolescence that suggests that vulnerabilities have a finite life cycle. It is important to quantify these phenomena.

Previous research has studied the software vulnerability lifecycle [2], the phases through which a vulnerability can progress. While later research has refined the lifecycle and quantified the length of some phases, most studies have focused on the relationship between when vulnerabilities are publicly disclosed and the development of patches or exploits [3, 7, 23]. A few studies have examined the timeframe from discovery to disclosure [1, 7]. Several studies have also analyzed other factors that may affect the lifecycle, including the effect of vulnerability severity on the time required for vendors to release patches, disparities in open versus closed-source vulnerability disclosure and patching [23], and the effect of code familiarity and reuse in the discovery of vulnerabilities in newly released software [4]. However, there has not been much effort to determine the total lifespan of a vulnerability from when it is created to when it is disclosed and patched. This is significant considering the extent of code reuse and legacy (old) software.

In part this is because of the many subtleties involved in the lifecycle milestones. A “honeymoon effect” occurs in which the interval from software release to the first vulnerability disclosure was longer than the interval separating disclosure of the second successive vulnerability, as seen in 62% of 30,000 vulnerabilities [4]. It appears that hackers become increasingly familiar with code and progressively take less time to discover new vulnerabilities. Vendors will also appear to patch a vulnerability more quickly once it has been disclosed, by an average of 28 days versus 65 days for delayed disclosure for 420 vulnerabilities from the National Vulnerabilities Database [3]. Statistical hazard modeling showed that the probability that a patch would be released was 2.5 times higher after disclosure. However, while immediate disclosure raises public awareness and may motivate the vendor to release a patch quickly, it gives malicious actors more time to create and deploy exploits before a patch is available. Manual updates were identified as a major reason for patching delays for four Internet browsers [8]. Besides patch-update mechanisms, delays can also be due to user negligence [20].

Lifecycle analysis is important for cybersecurity [16], but it is especially important in preparing countermeasures for the more serious threats of cyberweapons [14]. Current Russian operations in Ukraine have involved many cyberweapons. More details of these applications of our work are in [13].

Obtaining data of the software vulnerability lifecycle

The software-vulnerability lifecycle is the sequence of events during a vulnerability's existence. Three delays in the cycle affect the perishability and obsolescence of an exploit: awareness delay, patching delay, and adaptation delay [24]. They correspond to risks to the attacker that an exploit will not achieve desired effects due to patching and other countermeasures. Most research has focused on the length of these periods and their relationship to when the exploit is available.

While disclosure and discovery are useful milestones to assess the timeliness of subsequent events such as patch release, they provide an incomplete model of the lifecycle of a vulnerability because they do not consider when it was created. This information can often be estimated using product or version release data for the software [4]. Using this information, we can assess the probability of disclosure based on the age of a vulnerability, providing greater understanding of its obsolescence.

Formalizing the software-vulnerability lifecycle model, four dates are important for a vulnerability: when it was created ($t_{creation}$), when it was publicly disclosed ($t_{disclosure}$), when a patch was released by the vendor (t_{patch}), and when an exploit was available for that vulnerability ($t_{exploit}$). The discovery date, $t_{discovery}$, was unavailable for all the databases we used. Data from sources can be correlated with common vulnerabilities and exposures identifiers (CVE IDs) to distinguish vulnerabilities, obtained from the National Vulnerabilities Database (NVD) of NIST [18], from which we obtained 68,667 vulnerabilities for the years 2018-2021. This dataset spanned 8,740 unique vendors and over 19,000 software product lines. Our analysis required data from individual vendors as well as manual data collection of some software information. To aid this, we limited data collection to vulnerabilities for the operating-system product lines of Windows, Apple (iOS, tvOS, watchOS, and MAC OS), Android, Google Chrome, and several Linux products (Red Hat, Fedora, Debian, Ubuntu, and the Linux Kernel). This yielded a dataset of 10,912 vulnerabilities with which to work.

Vulnerabilities are assigned a common vulnerability scoring system (CVSS) score by NIST measuring their severity using factors of the attack vector, complexity, level of privilege required, scope, and level of required user interaction [18]. Vulnerabilities are also assigned weakness enumerators, which place the vulnerability in a family with similar attributes such as memory-buffer errors or exposure of sensitive information. Each CVE ID also has "common platform enumerations" (CPEs) which identify products and versions affected by that CVE. We estimated the creation date for a vulnerability as the release date of the earliest affected version.

For $t_{creation}$ dates, we used the public release date for the earliest software version in the NVD containing the vulnerability. We did not include dates for pre-releases (beta releases) because they are part of software development. For $t_{disclosure}$ dates, we used the date when the vulnerability was published to the NVD. We col-

lected t_{patch} dates from vendor security bulletins or databases, and used the vendor's update release dates. Published security updates usually mention the CVEs corrected by the update. The vendors for the products in our dataset keep central repositories for their security bulletins. We extracted this information with Python's Scrapy module [25] which can navigate Web pages, and used the Selenium module [21] with Scrapy for dynamic Web pages.

We collected $t_{exploit}$ dates from the Exploit Database (https:\\www.exploit-db.com), a public database of exploits created for penetration testing. Its dates were those on which exploits were published; they do not include delays in discovery and reverse engineering that we would expect to precede disclosure if a malicious exploit were used against an unsuspecting target. The database contains approximately 50,000 exploits, some of which are tagged with CVE IDs to identify the vulnerabilities they exploit. Having trouble with its tool Searchsploit, we again used Scrapy to crawl the database and extract the CVE IDs and the dates exploits were published. We created a list of affected versions for each CVE ID and then searched the Internet manually to find the version-release dates. While many release dates were available through vendor websites, others were found in third-party sources such as blogs. If more than one date existed for a CVE ID, such as when more than one patch was created for vulnerabilities affecting multiple products, we used the earliest date.

Data processing

The JSON-format file from the NVD lists vulnerabilities and their attributes, including their CVE IDs, CVSS scores, and product identifications. We developed a Python script that uses Python's JSON module to load the files of the NVD, parse them, extract the necessary data elements, generate the vulnerability index, and export the data to CSV files. The depth of the NVD files resulted in several layers of nested Python dictionaries, and we had to work down each layer to retrieve the desired data elements.

For each product and version listed for a product line, we searched the Internet for the corresponding release date. We started by searching vendor websites; some, like Microsoft, kept a complete list of release dates. Other vendor websites listed major releases but not minor releases such as an upgrade from 6.0 to 6.0.1. When the vendor website was inadequate, we searched blogs, bulletins, and other sources of information. If subsequent versions were affected, we used the release date for the first version. Windows patch release dates were obtained from the Microsoft Security Response Center website (https://msrc.microsoft.com/update-guide). The exported CSV file included the vulnerabilities patched by each update. To get patch data for the other product lines, we used the Scrapy module to crawl the websites where vendors published their security bulletins.

For each bulletin, we extracted the date it was published (the patch release date) and any CVE IDs, then wrote this to a CSV file for each vendor. For Fedora, which has an open-source development and update system, we only used the date on which the build was declared stable as the patch date. Some websites were harder to crawl, like Redhat, which used JavaScript on its customer portal. To handle these cases, we used the Selenium Python module. For some websites, Scrapy erroneously treated some links as duplicates and filtered them out, so we changed the setting in those spiders to disable filtering.

To get dates that exploits were published, we extracted them with the associated vulnerabilities using the same process as for patch data, by using another spider to crawl the Exploit Database website. This spider had several problems. Initially, HTTP GET messages returned a code 403 (Forbidden) status code, which we corrected by adding Mozilla Firefox headers to the settings for our spider [15]. Once we got it crawling, the Web server sometimes spontaneously ended the TCP connection, probably because we looked like a denial of service (DoS) attack. We fixed this problem by adjusting the spider settings to delay successive requests by one second. The Web server would sometimes redirect us back to the main page instead, which we corrected by using full page links. Some page links were identical except for the exploit serial numbers which were generally sequential integers, so we just retrieved pages in numerical sequence, which worked because the spider would skip over pages not found.

Following data extraction, we observed multiple entries with the same vulnerability ID, most likely due to multiple patches correcting the same vulnerability for different products. We removed the later such instances. Lastly we copied the creation, patch, and exploit dates to the CVE index, giving us $t_{creation}$, t_{patch}, and $t_{exploit}$ by using another Python script we wrote. Because the creation dates were assigned by product/version groupings and not by CVE ID, our Python script checked that each product line and product/version grouping was listed for each vulnerability in the CVE index. If it was, and a $t_{creation}$ value was not yet assigned, the creation date for that software version was assigned as $t_{creation}$ for that vulnerability. If a $t_{creation}$ value was already assigned, for those vulnerabilities in more than one product line or in more than one version in a product line, the earlier date was used.

Results

We collected 10,912 operating-system vulnerabilities from the NVD, all with publishing dates ($t_{disclosure}$). Of these, we found $t_{creation}$ for 7,893 vulnerabilities, t_{patch} for 8,860 vulnerabilities, and $t_{exploit}$ for 322 vulnerabilities. The four durations we were interested in were the time to disclosure(Δ_{cd}), the time to patch (Δ_{dp}), the time to exploit (Δ_{de}), and the longevity (Δ_{cp}). Δ_{cd} is the duration from $t_{creation}$ until $t_{disclosure}$, while Δ_{dp} and Δ_{de} are the durations from disclosure until patches

and exploits were published. Δ_{cp} represents the total lifespan of a vulnerability, from $t_{creation}$ to t_{patch}. Table 1 contains the statistics for each period for the dataset.

We first observed Δ_{cp} using the subset of vulnerabilities for which we had both $t_{creation}$ and t_{patch}; the median lifespan was 3.86 years (1,410 days). We also found a high degree of dispersion among the values: The bottom quartile was patched in less than two years (665 days), while the top quartile was patched after more than 7.1 years (2,600 days). The cumulative distribution function (CDF) or integral (Fig. 1) shows that when an operating system was first released, over 70% of vulnerabilities were patched within the first five years (1,825 days). After that, the rate of patching slowed considerably.

Table 1: Overall vulnerability statistics

	Vulnerabilities Observed	**Median (days)**	**25th Percentile**	**75th Percentile**
Time to Disclose (Δ_{cd}) ($t_{disclosure}$ - $t_{creation}$)	7893	1364	708	2371
Time to Patch (Δ_{dp}) (t_{patch} - $t_{disclosure}$)	8860	-1	-14	8
Time to Exploit (Δ_{de}) ($t_{exploit}$ - $t_{disclosure}$)	322	1	-23.75	7
Longevity (Δ_{cp}) (t_{patch} - $t_{creation}$)	7275	1410	665	2600

Interestingly, for 76 vulnerabilities the patch was released before the product containing the vulnerability. Further examination of those vulnerabilities indicated that the patch was sometimes released by a developer to fix third-party software before the release of the affected operating-system version; the operating system was probably released with old, unpatched software. In other cases, multiple operating systems were affected by the vulnerability, but the date of earliest affected version was unavailable. Other cases could not be easily explained from the data in the NVD and in security bulletins from the operating-system vendor; several of these cases involved the Fedora operating system.

To better model the probability of a vulnerability reaching a specified longevity, we used the Python Lifelines package [6] to fit several survival functions for the dataset. The survival function $S(t)$ is the probability that an event will occur after a specified time t, and it is commonly used in clinical trials in medicine. Survival analysis also permitted us to include censored data such as unpatched vulnerabilities [10]. We tried the Kaplan-Meier curve first because it is a nonparametric function that did not require us to fit a distribution to the data. The curve is $\hat{S}(t) = \prod_{i:t_i \leq t}(1 - \frac{d_i}{n_i})$, where d_i represents the number of events that have occurred and n_i the number of surviving elements at time t_i [11].

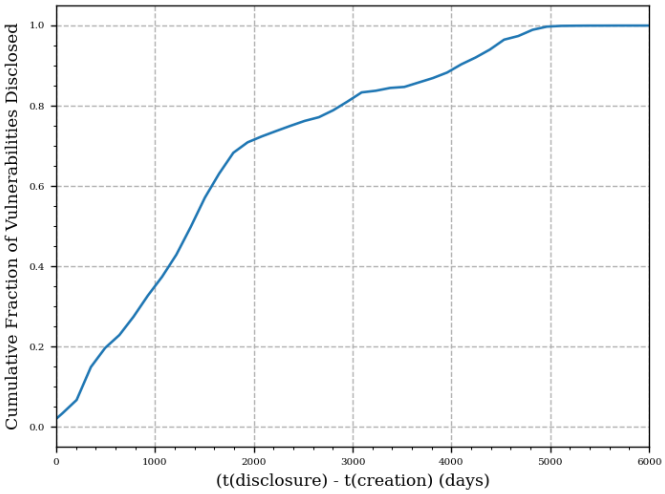

Fig. 1: Histogram and CDF of vulnerability longevity (Δ_{cp})

The median survival time was 1,485 days, with a 95 percent confidence interval of 1467 to 1517. That is higher than the 1,410 days observed including the unpatched data, suggesting that several vulnerabilities have remained unpatched for years, possibly because the software is no longer supported. The survival function showed a steep drop in the survival rate out to approximately 2,000 days; the probability a vulnerability will remain unpatched past this point is less than 0.35. Then the survival rate declined more slowly; at ten years (3,650 days), the probability of survival is still 0.18. This is consistent with the CDF in Fig. 1. We also fit a Weibull distribution based on the shape of the CDF (Fig. 3). It had a survival function $S(t) = e^{(-(\frac{t}{\eta})^{\beta})}$ with η = 2033.55, β = 1.26, and γ = 0 [19].

We also modeled the durations of three phases: time to disclose (Δ_{cd}, $t_{disclsoure}$ - $t_{creation}$), time to patch (Δ_{dp}, t_{patch} - $t_{disclosure}$), and time to exploit (Δ_{de}, $t_{exploit}$ - $t_{disclosure}$). We observed a median time to disclosure of 1,364 days (3.74 years). As with the longevity results, the standard deviation of 1,334 days indicated high variability in the dataset. The bottom quartile of vulnerabilities would be disclosed in less than two years (708 days), while the top quartile would require more than 6.5 years (2,371 days).

Time to patch after disclosure Δ_{dp} had a median of -1 days, and only 35.68 percent of patches were released after disclosure. This suggests that white-hat hackers and security researchers often report vulnerabilities to vendors before disclosing them, or do "responsible disclosure" [22]. Some vendors provide bug bounties or other incentives to encourage this behavior, which gives them time to

develop a patch before the vulnerability is disclosed. Most patches were released within weeks of the date of disclosure; the range of the middle quartiles (25th - 75th percentile) was [-14, 8] days. This concentration near the disclosure time is clear in the CDF (Fig. 2). However, the data becomes more widely dispersed as the time before or after disclosure increases, resulting in a standard deviation of 111.4 days.

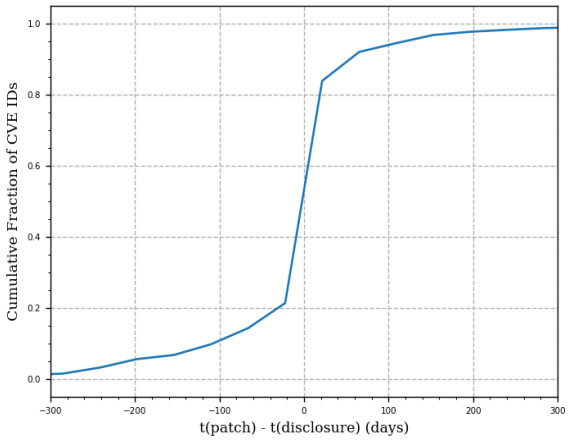

Fig. 2: Histogram and CDF for time to patch (Δ_{dp})

We only had 322 exploit dates for vulnerabilities in our data. 58 percent of exploits were released after their vulnerability had been disclosed. As with time to patch, the histogram for Δ_{de} showed that most exploits were published shortly after disclosure. Two reasons may explain this. First, the actors that populate the Exploit Database include many who do vulnerability research and penetration testing, white-hat actors willing to cooperate with vendors to privately disclose vulnerabilities. Second, many exploits are probably developed after disclosure of vulnerabilities with reverse engineering of patches [7].

We measured the effect of having a known exploit on vendor patching, using data in the Exploit Database. We assumed that the reaction from vendors to exploits published by white-hat actors and penetration testers would suggest their response to malicious exploits. Of 116 vulnerabilities with exploits released before disclosure ($\Delta_{de} < 0$), 110 were patched and six were unpatched. We saw that most vulnerabilities were patched before they were exploited; the median for Δ_{de} was -81 days while the median for Δ_{dp} was -101.5 days. For only 12 vulnerabilities was the exploit released before the patch ($\Delta_{de} < \Delta_{dp}$). All but one was disclosed within 20 days after the exploits were published.

We tried to determine how long it took vendors to react when a vulnerability was disclosed of which they were unaware . To measure this, we used the subset

of vulnerabilities where $\Delta_{dp} > 0$. We assumed that for this subset of vulnerabilities, coordination between white hats and vendors was unlikely because a vendor would not want a vulnerability to be disclosed until they could develop and release a patch. We narrowed our dataset to 3,161 vulnerabilities where $\Delta_{dp} > 0$. The median time to patch was 18 days, while the mean was 57 days, so some vulnerabilities took a long time to patch after they were disclosed. As with longevity, we fit survival functions to this dataset. The Weibull distribution was the best fit with $\beta = .70$, $\eta = 42.48$, and $\gamma = 0$. The median survival time using this distribution was 25.09 days, meaning a 50 percent chance a vulnerability will survive past this point. Unlike with longevity, this Weibull distribution model used only uncensored data, because we were more interested in understanding the time it took for vendors to release patches when they felt they needed to do so, rather than considering those that were not patched.

We also sought to determine any significant differences in the software vulnerability lifecycle based on the severity of a vulnerability. CVSS scores are segregated into four levels of severity: low (0-3.9), medium (4.0-6.9), high (7.0-8.9), and critical (9.0-10.0) [18]. For most phases of the software vulnerability lifecycle, the medians for each level were similar, except for Δ_{de}, for which we had less data. The CDFs for longevity are shown in Fig. 3; for approximately the first 1,500 days, the CDFs track closely together, with over half being patched during that period, then they begin to diverge slightly. Apparently high-severity vulnerabilities were patched more slowly, and critical severity vulnerabilities were patched more quickly than the others, although all four CDFs track closely to one another. After 1,500 days, we expected that more severe vulnerabilities would be discovered and patched faster due to the greater risk that they pose, but this suggests they are harder to find and patch.

When we compared patching behavior (Δ_{dp}) by CVSS severity, the statistics were like the overall longevity data: The datasets for each level had large standard deviations. The medians were all similar and close to zero, indicating that at least half of the vulnerabilities were patched on or before their disclosure. Fig. 4 shows the CDFs for Δ_{dp} at each severity level; little difference existed between them except for Low-severity vulnerabilities, for which a greater portion appeared to be patched before disclosure before falling below the other levels during the first 200 days after disclosure. However, that subset was small (233 vulnerabilities).

We also grouped our product lines into four groups for analysis: Windows, Linux distributions, Apple products, and Android. We found that Apple operating system vulnerabilities had the shortest longevity: The median Δ_{cp} was 737 days for Apple. Android was 909 days, Linux was 1,011 days, and Windows was over 10 years (3,654 days). Less than 40 percent of Windows vulnerabilities were patched after 2,500 days (6.8 years). In contrast, Apple and Android lacked any vulnerabilities whose lifespan exceeded 1,400 days. A possible reason that vulnerabilities for Apple and Android have shorter lifespans is that they release major operating system versions more often than their counterparts. For example, Apple's macOS 10.14 (Mojave) was released on September 24, 2018, but reached end-of-life in

late 2021, a lifespan of only three years [17]. Vendors are unlikely to disclose vulnerabilities for products after they reach end-of-life because security support and software updates are no longer provided. Even if the vulnerability affects versions currently in use, a vendor is probably less likely to mention legacy versions when disclosing vulnerabilities.

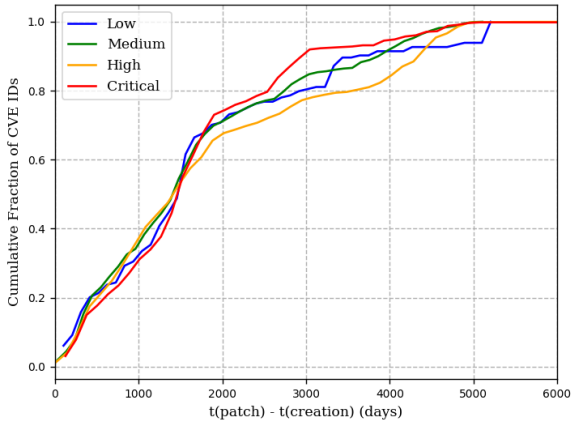

Fig. 3: Longevity CDFs by CVSS severity

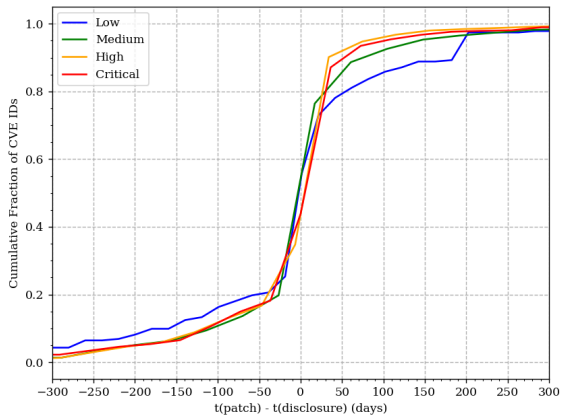

Fig. 4: CDF of time to patch by CVSS severity

We also studied variations in patching behavior (Δ_{dp}) between each operating system group. Apple, Android, and Windows each patched over 80 percent of

their vulnerabilities before their disclosure date. What is noteworthy about Windows is that out of 2,138 vulnerabilities in the Windows subset, all were patched on or before the date they were disclosed. In contrast, the Linux-based operating systems were much slower to patch their vulnerabilities. One reason could be that Linux vendors rely heavily on community development, and vulnerabilities must be disclosed to mobilize the developers to create a patch.

Conclusions

Our results provide both bad and good news about cybersecurity. The bad news is that we observed vulnerabilities which persisted for years without being patched, for which software producers were to blame. Vulnerabilities are not inevitable, but are the result of overly complex and poorly managed software engineering. If food production had a similar record in finding production mistakes, there would be much outrage. We did observe that most patches were released within days of vulnerability disclosure, suggesting close collaboration between third parties and vendors. The good news is that many vulnerabilities were patched before disclosure. This is likely not due to discoveries by the software producer because producers rarely test after release, and is not due to reports by users because users only rarely see vulnerabilities. This suggests that third parties play an important role in cybersecurity.

Acknowledgements

This work was supported by the Department of Energy through Idaho National Laboratories. Statements reflect the views of the authors and do not necessarily reflect those of the U.S. Government.

References

[1] Ablon, L., & Bogart, A. (2017). *Zero days, thousands of nights: The life and times of zero-day vulnerabilities and their exploits* (Report No. RR1751). RAND. https://www.rand.org/pubs/research reports/RR1751.html

[2] Arbaugh, W., Fithen, W., & McHugh, J. (2000). Windows of vulnerability: A case study analysis. *Computer, 33*(12), 52–59. http://dx.doi.org/10.1109/2.889093

[3] Arora, A., Krishnan, R., Telang, R., & Yang, Y. (2010). An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. *Information Systems Research, 21*(1), 115–132. http://www.jstor.org/stable/23015522

[4] Clark, S.; Frei, S.; Blaze, M.; & Smith, J. (2010). Familiarity breeds contempt: the honeymoon effect and the role of legacy code in zero-day vulnerabilities. *Proceedings of the 26th*

Annual Computer Security Applications Conference (ACSAC '10), 251-260. https://doi.org/10.1145/1920261.1920299

[5] Dacey, R. (2003). *Information security effective patch management is critical to mitigating software vulnerabilities* (GAO-03-1138T). Government Accountability Office. https://www.gao.gov/assets/gao-03-1138t.pdf

[6] Davidson-Pilon, (2019). Lifelines: survival analysis in Python. *Journal of Open Source Software, 4*(40), 1317, https://doi.org/10.21105/joss.01317

[7] Frei, S., May, M., Fiedler, U., & Plattner, B. (2006). Large-scale vulnerability analysis. *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense* (*LSAD '06*), *131-138*. https://doi.org/10.1145/1162666.1162671

[8] Frei, S., Dübendorfer, T., & Plattner, B. (2009). Firefox (in)security update dynamics exposed. *ACM SIGCOMM Computer Communication Review, 39*(1), 16-22. https://doi.org/10.1145/1496091.1496094

[9] Huntley, W. (2016). Strategic implications of offense and defense in cyberwar. *2016 49th Hawaii International Conference on System Sciences, 5588-5595.* http://dx.doi.org/10.1109/HICSS.2016.691

[10] Kleinbaum, D. & Klein, M. (2005). *Survival analysis: A self-learning text* (2nd ed.). Springer.

[11] Lewinson, E. (2020, August 17). *Introduction to survival analysis: the Kaplan-Meier estimator.* Towards Data Science. https://towardsdatascience.com/introduction-to-survival-analysis-the-kaplan-meier-estimator-94ec5812a97a

[12] Libicki, M., Ablon, L., Webb, A. (2015). *The defender's dilemma: Charting a course toward cybersecurity* (Report No. RR1024). Rand Corporation.

[13] Lidestri, M. (2022, June). *Incorporating perishability and obsolescence into cyberweapon scheduling*, M.S. thesis, U.S. Naval Postgraduate School.

[14] Liles, S., & Poremski, E. (2015). Fusion of malware and weapons taxonomies for analysis. *Journal of Information Warfare, 14(*1), 75–83. https://www.jstor.org/stable/26487520

[15] Mamka, A. (2016, August 29). How to solve 403 error in scrapy. Stack Overflow. https://stackoverflow.com/questions/39202058/how-to-solve-403-error-in-scrapy

[16] Metrick, K., Semrau, J., & Sadayappan, S. (2020, April 13). Think fast: Time between disclosure, patch release and vulnerability exploitation — Intelligence for vulnerability management, part two. *Mandiant*. https://www.mandiant.com/resources/time-between-disclosure-patch-release-and-vulnerability-exploitation

[17] Ng, D. (2022, March 7). *macOS 10.14 Mojave - end of support*. University of California – San Francisco. https://it.ucsf.edu/news/macos-1014-mojave-end-support

[18] NIST (National Institute of Standards and Technology) (2022). *The National Vulnerability Database* [Data set]. https://nvd.nist.gov/vuln/data-feeds

[19] Reliasoft. (2002, April). Characteristics of the Weibull Distribution. Reliability Hotwire (14). https://weibull.com/hotwire/issue14/relbasics14.htm

[20] Sarabi, A., Zhu, Z., Xiao, C., Liu, M., & Dumitraş, T. (2017). Patch me if you can: A study on the effects of individual user behavior on the end-host vulnerability state. In *Passive and Active Measurement* (pp. 113–125). Springer International Publishing. https://doi.org/10.1007/978-3-319-54328-4_9

[21] The Selenium Project (2022). Selenium [computer software]. Software Freedom Conservancy. https://www.selenium.dev/

[22] Sen, R., Choobineh, J., Kumar, S. (2020). Determinants of software vulnerability disclosure timing. *Production and Operations Management, 29*(11), 2532-2552. https://onlinelibrary.wiley.com/doi/abs/10.1111/poms.13120

[23] Shahzad, M., Shafiq, M. Z., & Liu, A. X. (2020). Large scale characterization of software vulnerability life cycles. *IEEE Transactions on Dependable and Secure Computing, 17*(4), 730–744. https://doi.org/10.1109/TDSC.2019.2893950

[24] Smeets, M. (2018). A matter of time: On the transitory nature of cyberweapons. *Journal of Strategic Studies, 41*(1-2), 6–32. https://doi.org/10.1080/01402390.2017.1288107

[25] Zyte (2022). Scrapy [computer software]. Zyte. https://scrapy.org/