Computer Science and Information Systems 14(2):537–555 DOI: 10.2298/CSIS161010010R

Modeling the Delivery of Security Advisories and CVEs

Jukka Ruohonen[1] and Sami Hyrynsalmi[1,2] and Ville Leppänen[1]

[1] Department of Information Technology, University of Turku, FI-20014 Turun yliopisto, Finland
[2] Pori Department, Tampere University of Technology, P.O. Box 300, FI-28101 Pori, Finland
{juanruo, sthyry, ville.leppanen}@utu.fi

Abstract. This empirical paper models three structural factors that are hypothesized to affect the turnaround times between the publication of security advisories and Common Vulnerabilities and Exposures (CVEs). The three structural factors are: (i) software product age at the time of advisory release; (ii) severity of vulnerabilities coordinated; and (iii) amounts of CVEs referenced in advisories. Although all three factors are observed to provide only limited information for statistically predicting the turnaround times in a dataset comprised of Microsoft, openSUSE, and Ubuntu operating system products, the paper outlines new research directions for better understanding the current problems related to vulnerability coordination.

Keywords: security patching, vulnerability life cycle, negative result

1. Introduction

This empirical paper examines turnaround times between the publication of security advisories and CVE identifiers.[1] More specifically, the empirical quantity of interest is defined as the time difference (in days) between a CVE publication in the National Vulnerability Database (NVD) and the publication of an associated, CVE-referenced security advisory.

In general, these "advisory-CVE turnaround times" are analogous to many common empirical software engineering concepts and metrics, including, but not limited to, "time-between-failures" [15], "time-between-commits" [33], "time-between-disclosures" [17], and "problem resolution interval" [24], or the analogous "defect resolution time" [4]. In the scholarly software vulnerability research, these concepts resonate with vulnerability life cycle (VLC) modeling with its general goal of tracing vulnerabilities through such life cycle abstractions as commits, defects, discoveries, disclosures, patches, publications, and intrusions [2, 18, 36, 37]. Although the general research background is thus well-founded, limited attention has been given for the role of security advisories in VLC modeling.

In addition to filling this gap in the literature, this paper introduces a new, previously unexplored metric for VLC modeling: the age of software products at the time of security advisory releases. This "age-metric" is operationalized in the opening Section 2 together with a brief motivation of the VLC background. For putting the elaborated life cycle variables into work, a dataset with 46 operating system products is examined in Section 3. Estimation is carried with a conventional Poisson regression model. Conclusions and future research directions are presented in the final Section 4.

[1] This paper is a rewritten and extended version of an earlier conference paper [34] presented at the 6th International Workshop on Information Systems Security Engineering (WISSE 2016).

2. Background

The forthcoming discussion briefly paints the basic analytical canvas related to vulnerability life cycle modeling (Section 2.1). After this motivation, VLC modeling is discussed in Section 2.2 in preparation for operationalization and measurement details in Section 2.3.

2.1. Vulnerability Life Cycles

Life cycle thinking posits the evolution of a phenomenon from its birth to its death. When applied to VLCs, this characterization contains a paradox. Although vulnerabilities are given birth in software repositories [25, 26], vulnerabilities seldom die in the sense that there would not be affected products even decades after the release of the products. Therefore, conventional VLC modeling often ends to events at which tickets have been closed in bug tracking systems, patches have arrived for customers, patches have been tested, security advisories have been published, CVEs have been coordinated, signatures have been written for intrusion detection and prevention systems, and so forth. After these activities, there is not much that can be done for helping consumers running unpatched products.

As illustrated in Fig. 1, different analytical states can be stated for vulnerabilities before they have seen the remediation phase. It should be noted that not all arcs are drawn and directions are omitted because the states are not consistent for presentation purposes; for instance, the last two states, publication and patching, often switch places in practice, while an unknown vulnerability can be discovered, a secret can be disclosed, and so forth. Nevertheless, many of these analytical states deliver also the basic security risk viewpoint.

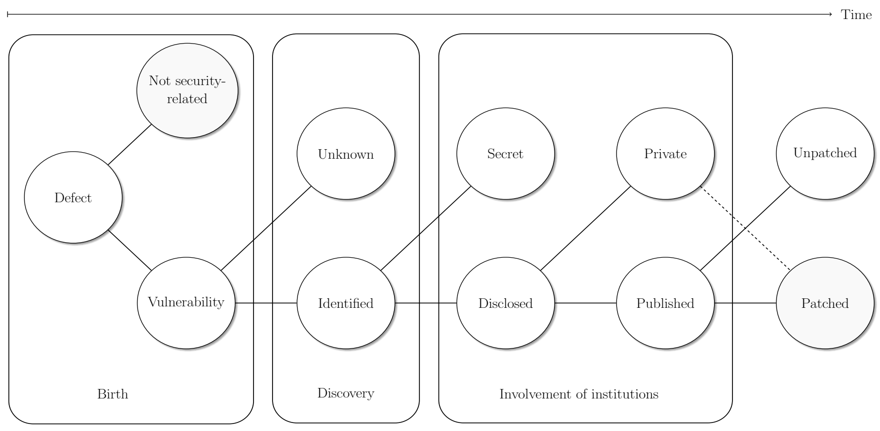

Fig. 1. Vulnerability Life Cycle States (basic idea with adjustments from [36])

After an actor has discovered a vulnerability by identifying a defect with security implications, the vulnerability in question can be thought to enter to the state of discovery. If the actor subsequently decides to keep the discovery as a secret, the security bug would be generally classified as a "zero-day" vulnerability according to the current jargon. Although these zero-day cases supposedly continue to expose some of the greatest security risks, the present interest relates to the more "mundane states" within and after the third rectangle in Fig. 1. This focus on the publication and patching states is no less important from a software engineering perspective. Because the current *de facto* practice is to use CVEs for identifying issues in security advisories, the publication state also involves coordination between vendors and the "disclosure institutions" [30] who are responsible for CVE tracking. Consequently, in terms of continuous software engineering practices, such as the so-called lean model, unnecessary time delays that may occur between these activities are a clear manifestation of waiting, and, thus, "waste" (cf. [14]). The optimal turnaround time from identification to patching should be short when all actors are benign professionals, of course, although, in practice, the delays can be considerably long.

In recent interesting work, analytical focus has been placed on the per-vulnerability time differences at an individual "within-state" level, that is, a subtraction "Disclosure$_{i+1}$ − Disclosure$_i$" can be used for modeling software engineering work of individual engineers [17]. This paper extends the same general idea for observing the "between-state" time lags in terms per-vulnerability "Advisory$_i$ − CVE$_i$" time differences. Given the overall optimum of having short time lags from the initial discovery to the publication and patching states, these differences contribute to the overall life cycle turnaround times.

2.2. Setup

In this paper, the analytical interest relates to the time difference

$$\begin{aligned} z_i &= \textit{Time of advisory} - \textit{Time of publication} \\ &= \tau_1 - \tau_0, \quad \text{given } i \text{ and } \; z_i \in (-\infty, \infty), \end{aligned} \tag{1}$$

and where τ_1 refers to the date and time at which an operating system software vendor released a security advisory that covered the i:th vulnerability, which was publicized with a CVE identifier at τ_0. Note that z_i is only theoretically restricted to be finite. If a vendor never patches a vulnerability, the life cycle of the vulnerability approaches infinity.

The scalar z_i can be understood as a simple efficiency metric for security patching, and, more accurately, for the associated release of security advisories. In general, a large positive value implies that a long time was required for a vendor to patch a vulnerability and communicate the information to users. When $z_i < 0$, a vendor handled a vulnerability before it was publicized at the infrastructure provided by MITRE and related institutions. Because all observed operating system vendors possess – either explicitly or via commercial sponsors – authorities for CVE assignments [23], these negative values are nothing special as such. For instance, Ubuntu released an advisory (USN-2628-1) for CVE-2015-4171 in 8th of June 2015, which was timestamped to the institutional databases two days later. Thus, an identifier was already available during the time of the advisory release, while the CVE publication was slightly delayed, possibly owing to additional processing and archiving work. Although numerical assessments with the Common Vulnerability Scoring System (CVSS) require additional manual work and further coordination between associated parties [36, 45], it should be remarked that z_i does not include the (often later) time points at which CVSS scores are published.

It should be also emphasized that a more traditional interest in empirical VLC modeling relates to the difference between τ_1 and the date at which information was first disclosed to a vendor or a third-party [3, 18, 41]. While such timelines are longer than the observed ones – disclosure must logically precede CVE publication, the analytical meaning remains more or less similar. The reason for preferring the state of publication (instead of the state of disclosure) relates to the well-known practical limitations imposed by the availability of robust vulnerability data [10, 19, 26]. In particular, disclosure dates – let alone discovery dates [17] – are only seldom known in practice [9, 36], and, hence, attachment to the publication state is necessary for maintaining a degree of conceptual rigor. In other words, this paper observes the later states in vulnerability life cycles, and as said, this focus is no less important than the length of vulnerability disclosure. From a coordination perspective [10, 16], the statistical optimum should be at $z_i = 0$, which implies a fully synchronized release of security advisories and CVEs. Systematically either positive or negative but large values of z_i indicate generally non-optimal coordination between operating system vendors, MITRE affiliates, NVD maintainers, and related parties.

2.3. Structural Factors

There are numerous methodological choices for modeling the scalar z_i. In this paper, the underlying modeling setup builds on two subsets, which capture the two scenarios illustrated in Fig. 2. Thus, let $y_1, \ldots, y_n$ denote a subset of timelines for which $z_i > 0$ for all i, that is, the cases that follow the route (b). Likewise, let $x_1, \ldots, x_n$ denote a subset of observations for which $z_i \leq 0$, implying the route (a) in the figure.

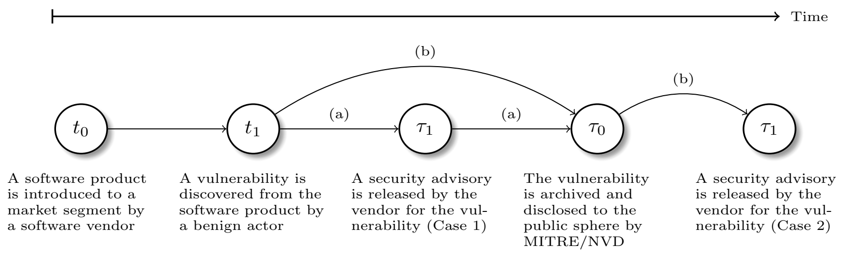
Time
(b)
(b)
(a)
(a)
t_0
t_1
τ_1
τ_0
τ_1
A software product is introduced to a market segment by a software vendor
A vulnerability is discovered from the software product by a benign actor
A security advisory is released by the vendor for the vulnerability (Case 1)
The vulnerability is archived and disclosed to the public sphere by MITRE/NVD
A security advisory is released by the vendor for the vulnerability (Case 2)

Fig. 2. Timelines for Advisory Releases (adopted from [34])

The interest is to examine three structural factors that are possibly statistically associated with the two timeline subsets. The term structural underlines that only meta-data measures are considered in the modeling, whereas, in reality, the lengths of the timelines in Fig. 2 are dependent on individual behavior, including the concrete software engineering work associated with disclosure, patching, writing of security advisories, communication with CVE-related institutions, and numerous related activities. While keeping this important point in mind, the three meta-data measures are used for positing two equations:

$$\begin{cases} f(\ y_i\ ) = \alpha_1 + \beta_{11}A_i + \beta_{12}S_i + \beta_{13}R_i \\ f(|x_i|) = \alpha_2 + \beta_{21}A_i + \beta_{22}S_i + \beta_{23}R_i \end{cases}, \tag{2}$$

where $f(w) = w$ for the time being, $y_i > 0$ and $x_i \leq 0$ for all i, $\{\alpha_1, \alpha_2, \beta_{11}, \ldots, \beta_{23}\}$ is a set of regression coefficients, S_i denotes the *severity* of the i:th CVE-referenced vulnerability, R_i is the cumulative amount of per-release security advisory *references* that were made to the same CVE identifier, and, finally,

$$A_i = \tau_1 - t_0, \quad A_i \geq 0, \tag{3}$$

approximates the *age* of a given operating system product at the time of the corresponding security advisory release. The general assumption is that $\beta_{11} \neq \beta_{12} \neq \cdots \neq \beta_{23} \neq 0$, meaning that the three structural factors can generally help at predicting the turnaround times. Although it is difficult to speculate about the signs and magnitudes of the coefficients, a few exploratory remarks can be made for motivating the statistical modeling.

The variable A_i provides a relatively straightforward hypothesis related to the age of operating system products [34]. Accordingly, and given a linearity assumption, when the age of a product increases by one day, particularly the mean length of the positive turnaround times, y_i, should increase by β_{11}, all other things being equal. Thus, older products would be generally more difficult to patch and coordinate, which would lead to expect that also $\beta_{21} < 0$ when modeling $|x_i|$. When only fixed software life cycles and publicly disclosed vulnerabilities are observed, it should be further remarked that the value of A_i is always non-negative, meaning that security patching only applies to products that have been released. A case $A_i = 0$ implies that a vulnerability was patched already during a product's release date – during the very first day of the product's life cycle. In general, however, the values $A_1, \ldots, A_n$ should be relatively large due to the so-called "S-curves" [22, 32] and "honeymoon effects" [11, 12]. That is, new software releases tend to enjoy short grace periods before the first vulnerabilities are discovered.

Following existing research [3, 22, 35], the severity variable $S_i \in [0, 10]$ is attached to the (aggregated) base CVSS score; the higher the value, the more severe the i:th vulnerability. Thus, also this variable provides a relatively logical hypothesis: severe vulnerabilities should be coordinated faster than more mundane vulnerabilities, which would lead to expect that $\beta_{12} < 0$ and $\beta_{22} < 0$. Although there are some existing empirical observations along these lines [40], the CVSS scoring system tends to provide only limited statistical variability between vulnerabilities [1], which subsequently undermines the empirical plausibility of the assumption. Furthermore, S_i is included to (2) irrespective of the date and time at which the CVSS scores were available from the CVE-processing institutions. In the context of security patching, the point is particularly important: if CVSS scores were not available at the time of patching, it is logically impossible to account for this information at the time when security advisories are written. Given these concerns, also the severity variable is included as a statistical control variable.

The variable R_i has been hypothesized to proxy security "patching quality" (see [40]; and references therein). From a software engineering perspective, the operationalization is arguably too coarse for rigorously evaluating the release and coordination strategies between advisories and CVEs. For instance, some vendors (such as Apple) have released large patch sets for large number of CVE-referenced vulnerabilities. Although such strategies should be visible via z_i, it is difficult to make the theoretical leap to quality without more fine-grained data and related background materials. Thus, also the last variable enters as a statistical control variable without prior expectations about the statistical effects.

3. Analysis

The empirical analysis operates with a dataset comprised of 46 operating system releases from three vendors. After introducing this dataset, the forthcoming discussion proceeds to briefly elaborate the statistical estimation strategy. Dissemination of the results follows.

3.1. Data

The empirical sample covers operating system releases of Microsoft Windows, openSUSE, and Ubuntu Linux (see Table 1). The case selection satisfies some desirable data collection properties: (a) open source software is included, and all observed products (b) have (and have had) a broad and loyal user base as well as (c) a large population of publicly disclosed vulnerabilities, which both allow presuming that (d) the products have been frequent targets of attacks and exploitation attempts [11]. In terms of operationalization, the timestamp τ_1 in (1) is fixed the security advisory release dates, whereas τ_0 is attached to the publication date at NVD. The time resolution is in days. Thus, it is plausible to assume (but not verify) that the released advisories have corresponded with availability of patches from download services.

Finally, the age variable A_i in (3) is computed with respect to the release dates of the observed operating system products. These products are listed in Fig. 3 (Microsoft), Fig. 4 (openSUSE), and Fig. 5 (Ubuntu). At the time of the data collection in the summer of 2015, Windows Vista, 7, and 8 were still eligible for security patches from Microsoft. In contrast, only two openSUSE and Ubuntu products were maintained, which generally signifies the longer life cycle of the Microsoft Windows products.

Table 1. Data Sources

	Institutions	Vendors / products		
	NVD	Microsoft	openSUSE	Ubuntu
Data source(s)	[27]	[20, 21]	[28, 29, 39]	[7, 8]

Note (i) that only openSUSE is sampled, although the advisories released for the commercial SUSE often account also affected openSUSE releases. In general, (ii) the concrete collection from the sources mimics the guidelines for more comprehensive data collection from multiple sources [19], including vulnerability databases, bug tracking systems, and software source code repositories.

Three general remarks are warranted about the dataset.

1. The dataset is generally in accordance with previous observations regarding the overall slowness of open source vendors [18, 36, 37]. When the per-vendor frequency distributions of the differences in (1) are examined, it is clear that Microsoft has been

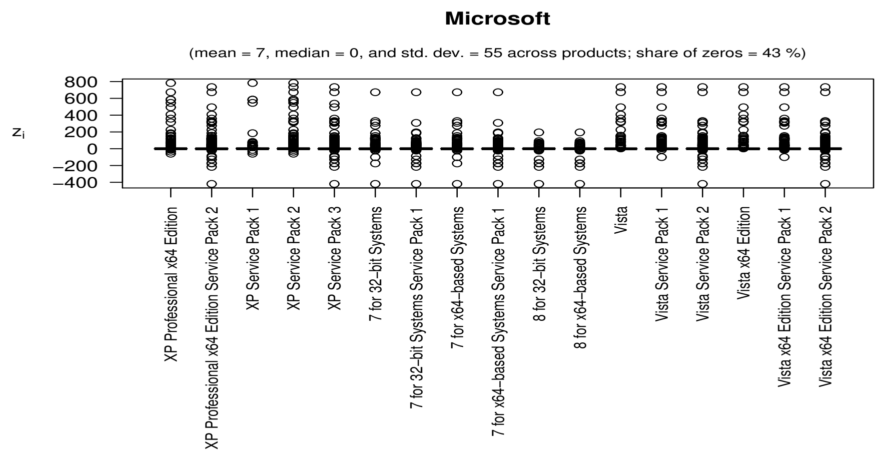

Fig. 3. Timelines for Microsoft Products

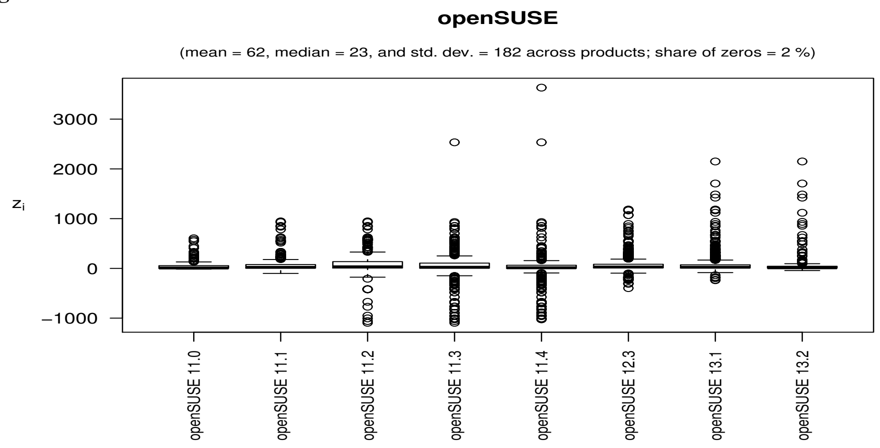

Fig. 4. Timelines for openSUSE Products

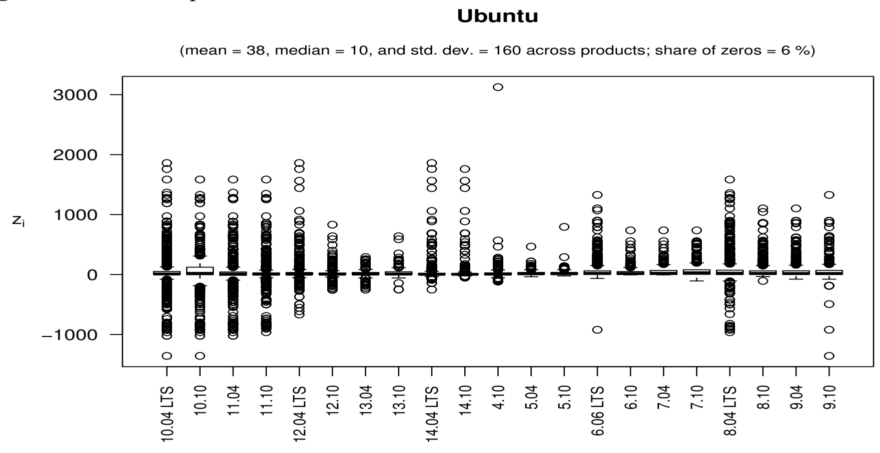

Fig. 5. Timelines for Ubuntu Products

faster than openSUSE and Ubuntu in coordinating the specific vulnerabilities that have affected the observed Microsoft Windows operating system releases. This general observation can be seen already by comparing the scales of the y-axis in Fig. 3 to the scales for the openSUSE and Ubuntu products shown in the subsequent two plots. As has been argued previously [13], it is difficult to pick a clear "winner" among open source operating system projects and associated vendors. The same applies to comparisons of the observed openSUSE and Ubuntu products.

2. A considerable amount of outliers is present, but mainly for the openSUSE and Ubuntu products. In general, much less dispersion is seen for the Microsoft products. In fact, Microsoft has patched as much as approximately 43 % of the observed vulnerabilities already during the same day when these were timestamped to NVD.
3. As indicated by the overall means, medians, and standard deviations across all per-vendor products reported in the subtitles of the three figures, most of the cases fall to the y_t subset timelines. Thus, roughly speaking, the route (b) in Fig. 2 has been more common. In other words, for the observed products, CVE publication has often preceded the publication of security advisories with back references.

It is difficult to speculate about the reasons explaining the outliers. What can be said, however, is that these are partially related to operationalization of the scalar z_i. In particular, (a) there are many-to-many references between advisories and CVEs, which leads to a notable operationalization problem. In this paper, the problem is approached by using the largest per-product advisory timestamp (the latest day) for each referenced CVE. Although also the reverse (the earliest dates) have been used [40], the present choice can be justified by maintaining that a given vulnerability was not entirely fixed and communicated to users until the last advisory released. On the other hand, for statistical analysis, the choice makes it a mystery why some openSUSE and Ubuntu advisories referenced old CVEs that had been available in NVD already 3,000 days ago. Although qualitative case studies would be required in this regard, it is also important to emphasize that (b) all three vendors support parallel products, and, hence, a single CVE-referenced vulnerability typically affects multiple products. The effect is pronounced in the case of Microsoft for which product variety has generally been larger within the observed product families.

3.2. Methods

The two equations in (2) both model a count data response variable against three covariates. That is, either y_i or $|x_i|$ counts days related to the i:th CVE publication timeline. Therefore, a classical Poisson regression model provides a sensible choice: the goal is to model the expected values (conditional means) of the two subset response variables. A standard "log-link" is readily available by defining $f(w) = \log(w)$ in (2). Thus, the expected values of the response variables are given by a *general specification* ($\mathcal{S}_1$):

$$\mathcal{S}_1 : \begin{cases} y_i &= \exp(\alpha_1 + \beta_{11}A_i + \beta_{12}S_i + \beta_{13}R_i) \\ |x_i| &= \exp(\alpha_2 + \beta_{21}A_i + \beta_{22}S_i + \beta_{23}R_i) \end{cases} , \tag{4}$$

meaning that the conditional mean is given for y_i by $e^{\alpha_1}e^{\beta_{11}A_i}e^{\beta_{12}S_i}e^{\beta_{13}R_i}$, and analogously for $|x_i|$. If $\beta_{11} = \beta_{12} = \beta_{13} = 0$, it follows that the expected value of y_i equals

$\exp(\alpha_1)$. Due to the issues related to model interpretation (see Section 2.3), this general "negative result assumption" provides a simple *null specification* ($\mathcal{S}_0$) in the form of

$$\mathcal{S}_0 : \begin{cases} y_i &= e^{\alpha_1} \\ |x_i| &= e^{\alpha_2} \end{cases} . \tag{5}$$

For assessing whether the three structural factors provide predictive power in general, the Baysian information criterion (BIC) values can be used for comparing $\mathcal{S}_0$ and $\mathcal{S}_1$. If the structural factors can provide hints for explaining the turnaround times, $\mathcal{S}_1$ should then attain noticeably lower values. For a further assessment, the magnitudes of the beta coefficients provide decent reality checks. (Unlike statistical significance; almost all coefficients are significant even at a $p < 0.001$ threshold, which, as such, does not arguably tell much due to reliability problems with vulnerability data, lack of random sampling, and related issues.) A classical issue with Poisson regression should be also noted.

The fundamental assumption is that the two subset timelines follow a Poisson distribution, which implies that the expected values of the responses should equal their variances. For applied work, a cautionary remark is therefore reserved regarding overdispersion, particularly in case there is an excess amount of zeros. Although a large amount of specifications exist for accounting the overdispersion problem [43], the recommendations for applied work remain generally inconclusive [5, 6]. On one hand, already Fig. 4 (openSUSE) and Fig. 5 (Ubuntu) allow deducing that the variances of y_i and $|x_i|$ do not equal their means for these two open source projects. Analogously, for the Microsoft Windows operating system products in Fig. 3, an analytical model selection diagram [6] would lead to prefer a negative binomial or related "zero-inflated" specification particularly due to the large amount of zeros. On the other hand, already the split into the two subset timelines (according $x_i \leq 0$ for all i) means that the same diagram presumably leads to different specifications for different products, which makes the comparison of results difficult. (It can be also remarked that the `glm.nb` function in the R package `MASS` [42] exhibits severe convergence problems for some products with the default settings.) Given these remarks, the specifications $\mathcal{S}_0$ and $\mathcal{S}_1$ are estimated as conventional Poisson regressions via the standard `glm` function in R. Therefore, some inaccuracies are to be expected, but the estimates should reveal further information about the effect of the beta coefficients.

Arguably, however, the overdispersion and related issues are estimation details when compared to choices made in the selection of a high-level estimation strategy. In particular, a so-called "fixed effects" (cross-sectional) strategy has been considered for empirical VLC modeling [3]. In contrast, (a) the subsequently reported results are computed by estimating the Poisson models equation-by-equation for the Microsoft, openSUSE and Ubuntu products separately. For presenting the results, the individually estimated specifications are evaluated by focusing on central tendency, dispersion, and potential presence of outliers. Furthermore, (b) it is assumed that observations within the subset timelines are independent from each other. In particular, (c) calendar time is not explicitly modeled, although the age variable (3) is supposed to proxy some of the calendar time effects. This restriction is problematic for some VLC questions [17], but it should not prevent the use of Poisson regression for studying the (independent) rate of occurrences. If dependencies are observed, one direction is provided for model refinement in further work.

3.3. Results

The BIC values are shown in Fig. 6 (Microsoft), Fig. 7 (openSUSE), and Fig. 8 (Ubuntu) for $\mathcal{S}_0$ and $\mathcal{S}_1$, estimated for each of the products individually. When interpreting the figures, it should be kept in mind that the plots are not comparable across the three vendors due to the unequal sample sizes. Although the effect may not be large due to the parallel product lines, comparisons across products should be also avoided because the releases affected do not remain constant for all vulnerabilities. Nonetheless, it is evident that the general specification $\mathcal{S}_1$ provides some information for predicting the subset timelines. For the clear majority of cases, the computed BIC values are smaller for $\mathcal{S}_1$ compared to $\mathcal{S}_0$. As a visual evaluation reveals, the differences are not large in magnitude, however.

The regression coefficient estimates are shown in Fig. 9 (Microsoft), Fig. 10 (openSUSE), and Fig. 11 (Ubuntu) using the general specification $\mathcal{S}_1$. The regression coefficient estimates $\hat{\beta}_{11}$, $\hat{\beta}_{12}$, and $\hat{\beta}_{13}$ are shown in the upper-row plots, whereas the lower plots summarize the coefficients $\hat{\beta}_{21}$, $\hat{\beta}_{22}$, and $\hat{\beta}_{23}$. Before interpreting the figures, it should be noted that the CVE-references variable R_i could not be included in all models (the amounts of these cases are marked with a NA symbol in the plots). As was noted in Section 3.1, there are often (but not always) many-to-many references between advisories and CVEs, but these relations do not necessarily exhibit any variance for some products, meaning that the inclusion of R_i would merely add another constant (and, hence, the variable must be omitted). As these cases apply only for a few outlying products, however, the plots can be disseminated by considering the coefficient magnitudes for each variable.

1. The coefficients for the age variable A_i are close to zero for all estimates. Although there are some outliers, a look at the y-axes is enough for revealing that the magnitudes are negligible. In other words, and as in previous examinations [34], the associated "age assertion" does not generally hold in the dataset.
2. Excluding only a few outliers, the coefficients $\hat{\beta}_{12}$ and $\hat{\beta}_{22}$ for the base CVSS scores are negative on average, as expected. The magnitudes are again small, however. The medians for the Microsoft (Fig. 9), openSUSE (Fig. 10), and Ubuntu (Fig. 11) products are -0.07, -0.06, and -0.07 in the y_i subset timelines, respectively. Thus, holding other variables constant, one unit increase in $\log(S_i)$ would decrease the timelines by a factor of $e^{-0.07}$ for the Microsoft products. To state the same via (2) with $f(w) = \log(w)$, one unit increase in the base CVSS scores would decrease the logarithm of the y_i subset timelines by a multiple of about 0.07.
3. For Microsoft and Ubuntu, the coefficients for R_i exhibit positive signs in the y_i subsets and negative signs in the x_i subsets on average. Because non-negative median coefficients are seen for openSUSE, the results are mixed for this variable. Furthermore, as discussed in Section 2.3, the interpretation is generally difficult for R_i.

Are the coefficient magnitudes sensible? As the coefficients are close to zero for the age variable, the question can be contemplated by focusing on the base CVSS scores. The effect of a day or few days does not seem illogical. Thus, severe vulnerabilities would tend to imply slightly faster coordination. Given the reliability, validity, and provenance problems [9, 17, 19, 26], a strong argument can be also formed by stating that the effects are largely noise. The argument is supported by a brief look at the fitted values.

The actual and predicted values for six products are shown for the y_i responses in Fig. 12 (Microsoft), Fig. 13 (openSUSE), and Fig. 14 (Ubuntu). For interpretation, note

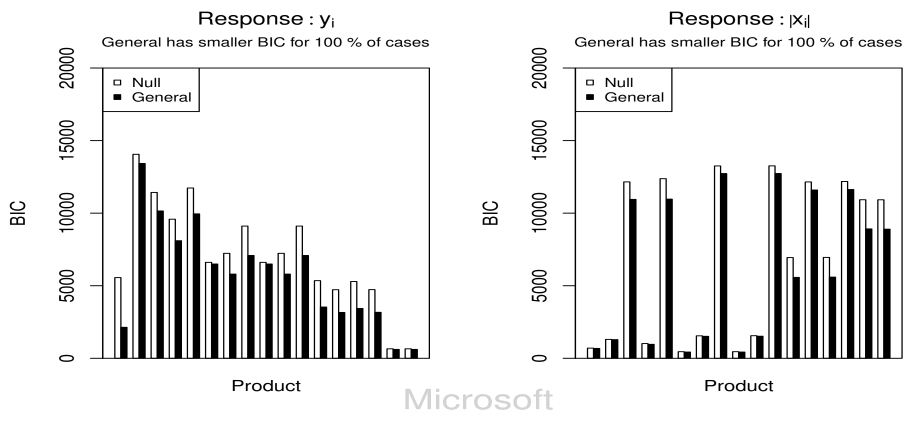

Fig. 6. BICs for Microsoft Products (Poisson, log-link, $\mathcal{S}_0$ and $\mathcal{S}_1$)

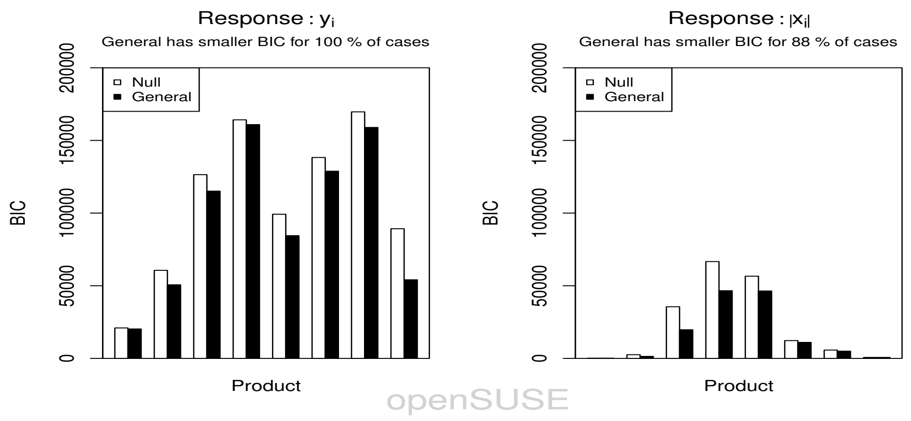

Fig. 7. BICs for openSUSE Products (Poisson, log-link, $\mathcal{S}_0$ and $\mathcal{S}_1$)

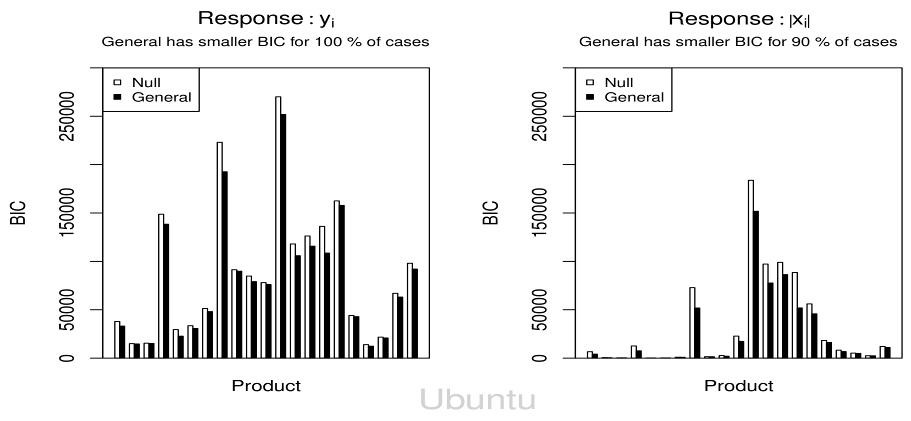

Fig. 8. BICs for Ubuntu Products (Poisson, log-link, $\mathcal{S}_0$ and $\mathcal{S}_1$)

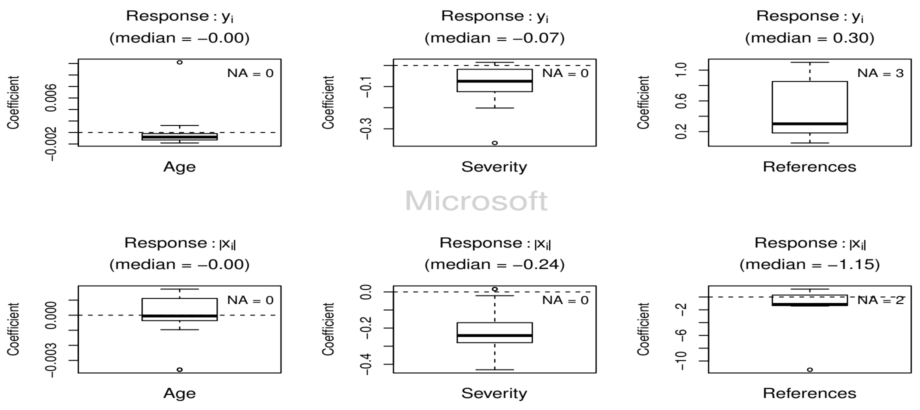

Fig. 9. Regression Coefficients for Microsoft Products (Poisson, log-link $\mathcal{S}_1$)

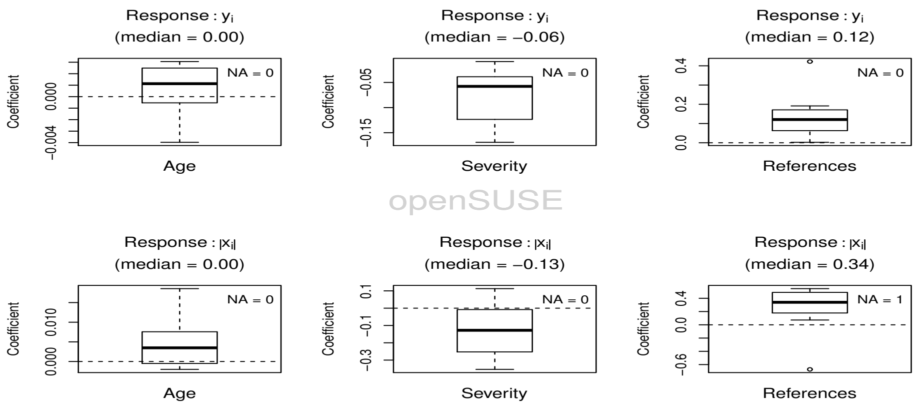

Fig. 10. Regression Coefficients for openSUSE Products (Poisson, log-link, $\mathcal{S}_1$)

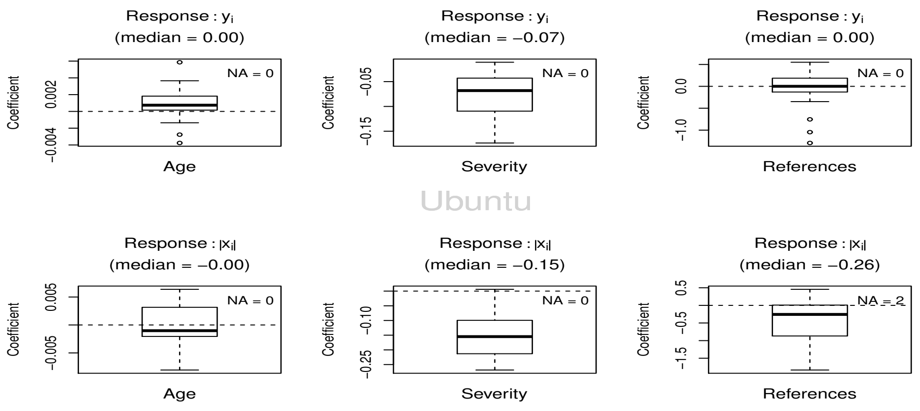

Fig. 11. Regression Coefficients for Ubuntu Products (Poisson, log-link, $\mathcal{S}_1$)

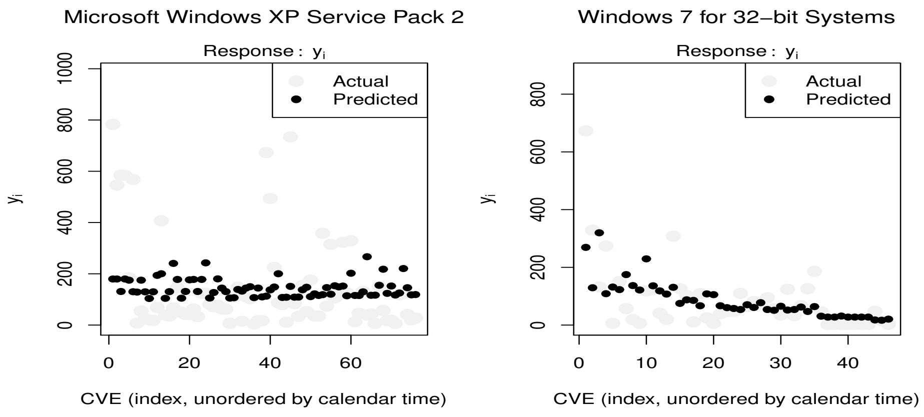

Fig. 12. Actual and Predicted Values for Two Microsoft Products (Poisson, log-link, $\mathcal{S}_1$)

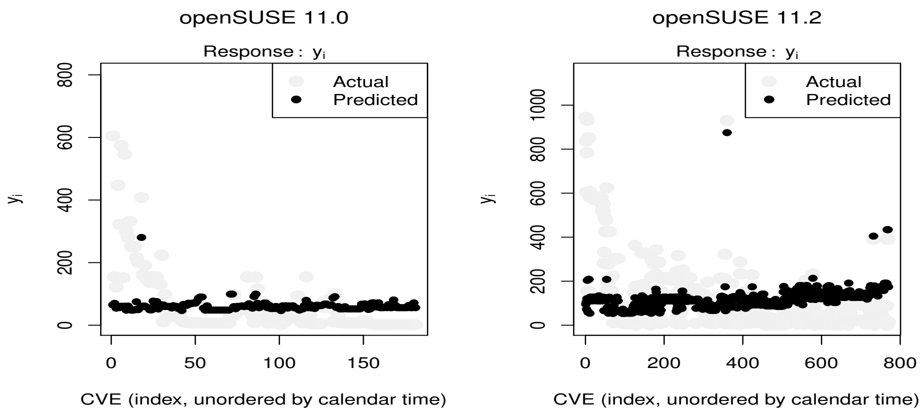

Fig. 13. Actual and Predicted Values for Two openSUSE Products (Poisson, log-link, $\mathcal{S}_1$)

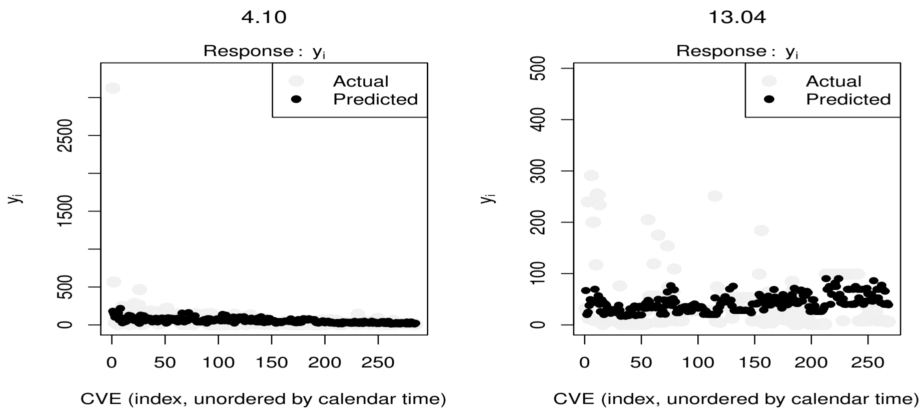

Fig. 14. Actual and Predicted Values for Two Ubuntu Products (Poisson, log-link, $\mathcal{S}_1$)

that x-axes (observations) have *not* been ordered according to CVE publication years and remaining identifiers. Thus, although there seem to be "trends", these clusters are spurious in terms of a calendar time interpretation. All in all, some plots indicate decent predictions for the conditional means. As visualized by the two openSUSE cases, however, some estimates are unable to capture the apparent clusters and outliers. Particularly for these products, something else largely explains the CVE publication turnaround times.

4. Discussion

The remainder of this paper summarizes the key results and future research directions.

4.1. Key Observations

To summarize the main findings, it can be started by noting that (a) coordination has been better for the observed Windows systems compared to the observed openSUSE and Ubuntu releases. Although care should be used when using vulnerability data for comparing vendors and products [10], this finding supports the existing empirical observations about the slowness of open source vendors [36, 37]. While it may well be that Microsoft is merely better at "gaming" the coordination processes (cf. [10]), it is likely that open source coordination exhibits some unique problems, including potential deficiencies in coordination between open source operating system vendors, between "upstream" developers and "downstream" distributors, and between related open source actors.

Irrespective of a product, (b) the age of a product at security advisory release times is not statistically associated with the coordination. In general, (c) severity fares a little better in this regard. The results for the base CVSS scores seem also relatively logical: it seems that severe vulnerabilities are coordinated slightly faster. For pursuing model refinements at this front, possible paths include unpacking the base scores to the individual CIA (confidentiality, integrity, availability) metrics [22, 40], conducting further randomized experiments [1], considering different weighting solutions [38], and, in general, conducting further empirical research for better understanding the potential role of severity in the coordination practices. By considering the usually severe "multi-vendor" vulnerabilities coordinated by US-CERT [31], for instance, it seems reasonable to hypothesize that severity might shed further light on the visible outliers observed in the dataset utilized.

However, the final key observation can be summarized by noting that (d) only modest statistical predictions can be computed for many of the observed products with the three structural factors alone. Something else is required for explaining the concrete reasons behind the turnaround times and software vulnerability coordination in general.

4.2. Future Directions

In general, (i) more thorough longitudinal modeling is required for accounting the long-run trends in calendar time. There are also some reasons to suspect that the estimates reported might be affected by non-random dependencies between the turnaround times. In this regard, classical reliability modeling [15, 44] might provide one path forward. Another option might be to consider different "sliding window variables" [22] for examining the longitudinal dimension. In other words, different "substates" could be added to the time intervals between τ_0 and τ_1 in Fig. 2.

More importantly, (ii) for evaluating how much "waste" [14] operating system vendors are accumulating during the publication state, a better understanding is required about the efficiency at the vendor-side in order for making reliable comparisons to the institutional setup, including the maintenance of NVD. Thus, it seems reasonable to argue that future VLC modeling should focus on backtracking to the initial birth state. Such tracing requires adopting software repository mining techniques, which also frame vulnerability life cycle modeling closer to the mainstream empirical software engineering research [25, 26]. However, it remains still unclear whether software repository mining can fully answer to a question about the factors affecting software vulnerability coordination.

Thus, (iii) another question relates to an argument that different "structural factors" derived from meta-data schemas cannot (or even should not) explain the advisory-CVE turnaround times. None of the vulnerability data warehouses were specifically designed to measure coordination and software engineering work. In other words, perhaps a more sensible path forward opens by considering individual [17], cultural [10], social, organizational, and institutional factors contributing to VLCs. The recent open source coordination efforts via the `oss-security` mailing list would offer a good case study in this regard. In addition to economic viewpoints [3, 22], explaining the big picture requires better understanding of the whole institutional setup used for handling and tracking software vulnerabilities. In many ways, CVE tracking is still today as it was in the 1990s according to current critics, plagued by different problems, ranging from cultural conflicts and language barriers [10] to poorly documented databases, data warehousing and database competition [9], governance issues, and limited funding.

4.3. Conclusion

This empirical paper modeled turnaround times between the publication of security advisories and CVE identifiers. A Poisson regression specification with three explanatory structural factors was used for examining a dataset comprised of nearly fifty operating system product releases. The three factors were: age (in terms of product launches), severity (base CVSS scores), and amount of CVE references in security advisories. Taken together, these three factors provide only limited information for statistically predicting the turnaround times. With this "negative result", the paper paves the way for further vulnerability life cycle research and practical attempts to improve CVE coordination, including the software engineering aspects related to security advisories and security patching.

Acknowledgments. The authors gratefully acknowledge Tekes – the Finnish Funding Agency for Innovation, DIMECC Oy, and the Cyber Trust research program for their support.

Bibliography

[1] Allodi, L., Massacci, F.: Comparing Vulnerability Severity and Exploits Using Case-Control Studies. ACM Transactions on Information and System Security 17(1), 1:1–1:20 (2014)

[2] Arbaugh, W.A., Fithen, W.L., McHugh, J.: Window of Vulnerability: A Case Study Analysis. Computer 32(12), 52–59 (2000)

[3] Arora, A., Forman, C., Nandkumar, A., Telang, R.: Competition and Patching of Security Vulnerabilities: An Empirical Analysis. Information Economics and Policy 22(2), 164–177 (2010)

[4] Assar, S., Borg, M., Pfahl, D.: Using Text Clustering to Predict Defect Resolution Time: A Conceptual Replication and an Evaluation of Prediction Accuracy. Empirical Software Engineering 21, 1437–1475 (2016)

[5] Berk, R., MacDonald, J.M.: Overdispersion and Poisson Regression. Journal of Quantitative Criminology 24(3), 269–284 (2008)

[6] Blevins, D.P., Tsang, E.W.K., Spain, S.M.: Count-Based Research in Management: Suggestions for Improvement. Organizational Research Methods 18(1), 47–69 (2015)

[7] Canonical, Ltd.: Releases (2015), available online in July 2015: `https://wiki.ubuntu.com/Releases`

[8] Canonical, Ltd.: Ubuntu Security Notices (2015), available online in March 2015: `http://www.ubuntu.com/usn/`

[9] Christey, S.: Open Letter on the Interpretation of ”Vulnerability Statistics” (2006), Appeared originally in *full-disclosure*, available online in July 2015: `http://www.gossamer-threads.com/lists/fulldisc/full-disclosure/40853`

[10] Christey, S., Martin, B.: Buying Into the Bias: Why Vulnerability Statisticks Suck. In: Proceedings of Black Hat 2013. Las Vegas (2013), available online in January 2017: `https://media.blackhat.com/us-13/US-13-Martin-Buying-Into-The-Bias-Why-\\Vulnerability-Statistics-Suck-Slides.pdf`

[11] Clark, S., Collis, M., Smith, J.M., Blaze, M.: Moving Targets: Security and Rapid-Release in Firefox. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS 2014). pp. 1256–1266. ACM, Scottsdale (2014)

[12] Clark, S., Frei, S., Blaze, M., Smith, J.: Familiarity Breeds Contempt: The Honeymoon Effect and the Role of Legacy Code in Zero-Day Vulnerabilities. In: Proceedings of the 26th Annual Computer Security Applications Conference (ASAC 2010). pp. 251–260. ACM, Austin, Texas (2010)

[13] Edge, J.: Ubuntu, Security Response, and Community Contributions (2008), Linux Weekly News (LWN). Available online in August 2015: `http://lwn.net/Articles/290156/`

[14] Fitzgerald, B., Stol, K.: Continuous Software Engineering: A Roadmap and Agenda. Journal of Systems and Software 123, 176–189 (2015)

[15] Goel, A.L., Okumoto, K.: Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures. IEEE Transactions on Reliability R-28(3), 206–211 (1979)

[16] Howison, J., Crowston, K.: Collaboration Through Open Superposition: A Theory of the Open Source Way. MIS Quarterly 38(1), 29–50 (2014)

[17] Johnson, P., Gorton, D., Langerström, R., Ekstedt, M.: Time Between Vulnerability Disclosures: A Measure of Software Product Vulnerability. Computers & Security 62, 278–295 (2016)

[18] Marconato, G.V., Nicomette, V., Kaâniche, M.: Security-Related Vulnerability Life Cycle Analysis. In: Proceedings of the 7th International Conference on Risk and Security of Internet and Systems (CRiSIS 2012). pp. 1–8. IEEE, Cork (2012)

[19] Massacci, F., Nguyen, V.H.: Which Is the Right Source for Vulnerability Studies? An Empirical Analysis on Mozilla Firefox. In: Proceedings of the 6th International Workshop on Security Measurements and Metrics (MetriSec 2010). pp. 4:1–4:8. ACM, Bolzano (2010)

[20] Microsoft, Inc.: Microsoft Security Bulletin Data (2015), available online in July 2015: `http://www.microsoft.com/en-us/download/details.aspx?id=36982`

[21] Microsoft, Inc.: Windows Life Cycle Fact Sheet (2015), available online in July 2015: `http://windows.microsoft.com/en-us/windows/lifecycle`

[22] Mitra, S., Ransbotham, S.: Information Disclosure and the Diffusion of Information Security Attacks. Information Systems Research 26(3), 565–584 (2015)

[23] MITRE, Inc.: CVE Numbering Authorites (as of February 2015) (2015), available online in June 2015: `https://cve.mitre.org/cve/cna.html`

[24] Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology 11(3), 309–346 (2002)

[25] Nguyen, V.H., Dashevskyi, S., Massacci, F.: An Automatic Method for Assessing the Versions Affected by a Vulnerability. Empirical Software Engineering 21(6), 2268–2297 (2015)

[26] Nguyen, V.H., Massacci, F.: The (Un)Reliability of NVD Vulnerability Versions Data: An Empirical Experiment on Google Chrome Vulnerabilities. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIACCS 2013). pp. 493–498. ACM (2013)

[27] NIST: NVD Data Feed and Product Integration (2015), National Institute of Standards and Technology (NIST), Annually Archived CVE Vulnerability Feeds: Security Related Software Flaws, NVD/CVE XML Feed with CVSS and CPE Mappings (Version 2.0). Available online in June 2015: `https://nvd.nist.gov/download.cfm`

[28] Novell, Inc. and others: openSUSE:Lifetime (2015), available online in July 2015: `https://en.opensuse.org/Lifetime`

[29] Novell, Inc. and others: openSUSE:Roadmap (2015), available online in July 2015: `https://en.opensuse.org/openSUSE:Roadmap`

[30] Ozment, A.: Improving Vulnerability Discovery Models: Problems with Definitions and Assumptions. In: Proceedings of the 2007 ACM Workshop on Quality of Protection (QoP 2007). pp. 6–11. ACM, Alexandria (2007)

[31] Ruohonen, J., Holvitie, J., Hyrynsalmi, S., Leppänen, V.: Exploring the Clustering of Software Vulnerability Disclosure Notifications Across Software Vendors. In: Proceedings of the 13th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2016). IEEE, Agadir (2016)

[32] Ruohonen, J., Hyrynsalmi, S., Leppänen, V.: The Sigmoidal Growth of Operating System Security Vulnerabilities: An Empirical Revisit. Computers & Security 55, 1–20 (2015)

[33] Ruohonen, J., Hyrynsalmi, S., Leppänen, V.: Time Series Trends in Software Evolution. Journal of Software: Evolution and Process 27(2), 990–1015 (2015)

[34] Ruohonen, J., Hyrynsalmi, S., Leppänen, V.: Software Vulnerability Life Cycles and the Age of Software Products: An Empirical Assertion with Operating System Products. In: Krogstie, J., Mouratidis, H., Su, J. (eds.) Proceedings of the CAiSE 2016 International Workshops, Lecture Notes in Business Information Processing (Volume 249). pp. 207–218. Springer, Ljubljana (2016)

[35] Ruohonen, J., Hyrynsalmi, S., Leppänen, V.: Trading Exploits Online: A Preliminary Case Study. In: Proceedings of the IEEE Tenth International Conference on Research Challenges in Information Science (RCIS 2016). pp. 1–12. IEEE, Grenoble (2016)

[36] Schryen, G.: Is Open Source Security a Myth? Communications of the ACM 54(5), 130–140 (2011)

[37] Shahzad, M., Shafiq, M.Z., Liu, A.X.: A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles. In: Proceedings of the 34th International Conference on Software Engineering (ICSE 2012). pp. 771–781. IEEE, Zurich (2012)

[38] Spanos, G., Angelis, L.: Impact Metrics of Security Vulnerabilities: Analysis and Weighing. Information Security Journal: A Global Perspective 24(1–3), 24–57 (2015)

[39] SUSE, LLC: Published SUSE Linux Security Updates by CVE Number (2015), available online in June 2015: `https://www.suse.com/security/cve/`

[40] Temizkan, O., Kumar, R.L., Park, S., Subramaniam, C.: Patch Release Behaviors of Software Vendors in Response to Vulnerabilities: An Empirical Analysis. Journal of Management of Information Systems 28(4), 305–337 (2012)

[41] Vache, G.: Vulnerability Analysis for a Quantitative Security Evaluation. In: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM 2009). pp. 526–534. IEEE, Orlando (2009)

[42] Venables, W.N., Ripley, B.D.: Modern Applied Statistics with S. Springer, Berlin, fourth edn. (2002)

[43] Xie, H., Tao, J., McHugo, G.J., Drake, R.: Comparing Statistical Methods for Analyzing Skewed Longitudinal Count Data With Many Zeros: An Example of Smoking Cessation. Journal of Substance Abuse Treatment 45(1), 99–108 (2013)

[44] Xie, M., Hong, G.Y., Wohlin, C.: A Practical Method for the Estimation of Software Reliability Growth in the Early Stage of Testing. In: Proceedings of the Eight International Symposium on Software Reliability Engineering (ISSRE 1997). pp. 116–123 . IEEE, Albuquerque (1997)

[45] Younis, A., Malaiya, Y.K., Ray, I.: Assessing Vulnerability Exploitability Risk Using Software Properties. Software Quality Journal 24(1), 159–202 (2015)

Jukka Ruohonen is a Ph.D. student at the University of Turku, Finland. His research interests cover software engineering, computer security, open source, applied statistics, and machine learning, among other things.

Sami Hyrynsalmi is a nerd who has always enjoyed working with programming and computers. After graduating as Master of Science in Technology in software engineering from University of Turku in 2009, he decided to focus on the real issues and started his doctoral dissertation work on mobile application ecosystems. After successfully defending his thesis in 2014, he has focused on various themes from software security to business ecosystems.

Ville Leppänen, Ph.D., works currently as a software engineering professor at the University of Turku, Finland. He has over 140 scientic publications. His research interests are related broadly to software engineering, ranging from software engineering methodologies, practices, and tools to security and quality issues, and to programming languages, parallelism, and algorithmic design topics.

Received: October 10, 2016; Accepted: April 10, 2017.