Software Vulnerability Life Cycles and the Age of Software Products: An Empirical Assertion with Operating System Products

Jukka Ruohonen, Sami Hyrynsalmi(✉), and Ville Leppänen

Department of Information Technology, University of Turku,
20014 Turun yliopisto, Finland
{juanruo,sthyry,ville.leppanen}@utu.fi

Abstract. This empirical paper examines whether the age of software products can explain the turnaround between the release of security advisories and the publication vulnerability information. Building on the theoretical rationale of vulnerability life cycle modeling, this assertion is examined with an empirical sample that covers operating system releases from Microsoft and two Linux vendors. Estimation is carried out with a linear regression model. The results indicate that the age of the observed Microsoft products does not affect the turnaround times, and only feeble statistical relationships are present for the examined Linux releases. With this negative result, the paper contributes to the vulnerability life cycle modeling research by presenting and rejecting one theoretically motivated and previously unexplored question. The rejection is also a positive result; there is no reason for users to fear that the turnaround times would significantly lengthen as operating system releases age.

Keywords: Security patching · Operating system · Negative result · Microsoft · Linux

1 Introduction

Vulnerability life cycle (VLC) modeling has been a popular methodology for understanding the longitudinal evolution of a vulnerability, from the initial birth at a version control system [18] through disclosure and patching [3,26] to the release of information to the public sphere, the availability of industrially developed of exploits, and the final loss of relevance [2]. Akin to software life cycle modeling [22], the primary interest in VLC modeling relates to the time lines between these and other theoretical stages, the most important ones of which are illustrated in Fig. 2. Different research questions emerge by shuffling the state diagram appropriately. For instance, in the worst case scenario, there are only three states: the birth, the discovery, and the development of an exploit;

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

J. Krogstie et al. (Eds.): CAiSE 2016, LNBIP 249, pp. 207–218, 2016.
DOI: 10.1007/978-3-319-39564-7_20

Fig. 1. The principal vulnerability life cycle states (adopted from [2])

a zero-day vulnerability remains to be a zero-day vulnerability, never reaching the states of disclosure, remediation, publication, and, over the years or decades, the eventual lack of applicable computer systems.

In this paper, the empirical phenomenon of interest is located in the time delays that occur between the patching and publication states; the case that is marked with a ⋆ symbol in Fig. 1. Ideally, these delays should be small, given the overall optimum of short vulnerability life cycles. Because the birth, discovery, and disclosure have already occurred, the observed delays are also directly controllable by the associated software vendors and the non-profit institutions that are partially responsible for the systematic release of information to the public sphere. Given this general efficiency rationale, the paper examines an assertion that the time delays are associated with the age of operating system products at the time of security advisory releases.

This assertion can be motivated by a common-sense software engineering reasoning: old and aging code bases are difficult to maintain, which leads to expect that also the handling of vulnerabilities takes longer for older products. The operating system context adds some weight to this reasoning. The code bases are complex, containing large amounts of low-level code that often requires special expertise to maintain. Because the discovery of new vulnerabilities tends to slow down as operating systems age [22], and as maintenance is often a necessary evil for software vendors, there may be a temptation to allocate insufficient resources for maintenance, which may then cause delays in security patching. Also the software life cycles are long. In fact, in some cases the life cycles are so long that the still maintained code bases may attain a status of legacy code. Although security has not been the driving force, analogous reasoning largely applies to the so-called rapid releases, which have been an increasingly widespread strategy for many products but apparently without notable quality and security consequences [7,9,12]. While rapid releases have been adopted also for Linux distributions, there has been also a trend to the opposite direction, as manifested by the so-called long-term support (LTS) releases. Then, by hypothesis, also security patching should vary according to the support periods.

The overall research strategy is exploratory, of course. In other words, numerous different factors influence the length between VLC states, ranging from technical aspects to organizational factors and communication obstacles. Nevertheless, on one hand: if the assertion holds, release engineering strategies could benefit from a footnote that the support period lengths may also increase the turnaround times in the coordination of vulnerabilities. On the other hand: if the assertion is not passed for notable operating system product releases, there are neither reasons to reserve space for such footnotes nor to complicate empirical VLC models with unimportant explanatory variables.

2 The Assertion

Analytically, the interest relates to the 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 Common Vulnerabilities and Exposures (CVEs) 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, Inc. and related non-profit institutions. All observed operating system vendors possess – either explicitly or via the commercial sponsors – authorities for CVE assignments, and, thus, 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 owning to additional processing and archiving work.

It should be also emphasized that a more traditional interest in empirical VLC modeling has related to the difference between τ_1 and the date at which information was first disclosed to a vendor or a third-party [3,27]. While such time lines are longer than the observed ones – disclosure must logically precede the publication of CVEs, the analytical meaning remains more or less similar (see Fig. 2). The reason to prefer the theoretical and conceptual state of publication, rather than the state of disclosure, relates to the well-known practical limitations imposed by the availability of robust data [14,18]. In particular, the date of disclosure is only seldom known in practice [23], and, hence, the attachment to the publication state is necessary to maintain a degree of theoretical

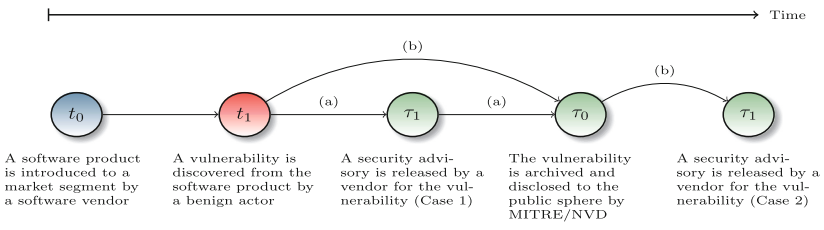

Fig. 2. A time line for security advisory releases (motivated by [6,11])

and conceptual rigor. In general, thus, this paper observes the later states in vulnerability life cycles (see Fig. 1). This observational focus is no less important than the length of vulnerability disclosure; systematically positive or negative but large values of z_i indicate that the coordination between operating system vendors, MITRE, and associated parties has been non-optimal.

Let $y_1, \ldots, y_n$ further denote a subset of time lines for which $z_i > 0$ for all i, that is, the cases that follow the route (b) in Fig. 2. Consider then that these positive turnaround times are linear functions of the *age*, A_i, of an operating system product at the time of patching. In other words,

$$y_i = \alpha + \beta A_i, \quad A_i = \tau_1 - t_0, \quad A_i \geq 0, \quad y_i > 0, \tag{2}$$

where α is a constant and β is a slope coefficient. When only fixed software life cycles and publicly disclosed vulnerabilities are observed, 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 honeymoon effect [7,8]. That is, new software releases tend to enjoy short grace periods before the first vulnerabilities are discovered.

Given the assumption of linearity, the sign of β is hypothesized to be positive: when the age of a product increases by one day, the mean length of the turnaround times, y_i, increase by β, all other things being equal. Thus, in general, aging operating system releases would be more difficult to patch. While the negative values from (1) have been also excluded in VLC modeling [26], in the present context it is relevant to augment the $\beta > 0$ assumption with a reverse formulation. That is, ψ can be asserted to be negative for an analogous relation

$$|x_i| = \alpha + \psi A_i, \quad x_i \leq 0, \tag{3}$$

given a subset $x_1, \ldots, x_n$ of observations for which $z_i \leq 0$. Because a modulus is used in (3), values $\psi < 0$ imply that aging products would tend to reduce the lead of vendors to the CVE-processing institutions; that is, the route (a) from τ_1 to τ_0 in Fig. 2 would reduce. These dual assumption constitute the *age assertion* of interest. Given that open source projects have been observed and argued to be slower in patching and vulnerability handling in general [23,24], it can be further expected that firing off the assertion is not universal among the contemporary population of operating systems products.

3 Evaluation

To evaluate the assertion, a dataset comprised of 46 operating system releases from three vendors is utilized by using a standard linear regression. While there is plenty of data (35,760 observations, to be precise), the data is not a representative sample from the contemporary operating system population. Thus, the specific term assertion carries a specific meaning: when the assertion is not

passed for the products of the market leader, Microsoft, Inc., it can be safely concluded that the assertion does not sufficiently characterize the contemporary operating system population. The evaluation criterion itself is not statistical.

3.1 Data

The empirical sample covers operating system releases of Microsoft Windows, openSUSE, and Ubuntu Linux. The case selection satisfies the desirable data collection conditions: (a) open source software is included, and all observed products (a) have (and have had) a broad and loyal user base as well as (c) a large population of publicly disclosed vulnerabilities, which both allow to assume that (d) the products have been frequent targets of attacks and exploitation attempts [7]. In terms of operationalization, τ_1 in (1) and Fig. 2 is fixed to the corresponding security advisory release dates (see Table 1), while τ_0 is attached to the publication date at the National Vulnerability Database (NVD). Given the temporal resolution of days, it is plausible to assume (but not verify) that the released advisories have corresponded with the availability of patches from the vendors' download services. Finally, the age variable A_i in (2) is computed with respect to the release dates of the observed operating system products listed in Table 2.

Table 1. Data sources

	Institutions	Vendors / products		
	NVD	Microsoft	openSUSE	Ubuntu
Source(s)	[19]	[15,16]	[20,21,25]	[4,5]

Note that only openSUSE advisories are sampled, although the advisories released for the commercial SUSE often account also the openSUSE releases that are affected.

The dataset is generally in accordance with previous observations [13]. When the per-vendor frequency distributions of the differences in (1) are examined, it is clear that Microsoft has been faster than openSUSE and Ubuntu in fixing the specific vulnerabilities that have affected the observed Microsoft Windows operating system releases (see Fig. 3). 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 (see Fig. 4). 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. All in all, these observations support the existing evidence that closed source vendors are faster than open source vendors [24]. As a prior-analysis expectation, the effects of β and ψ should thus be different for the Microsoft products.

Finally, it is important to emphasize that (a) all three vendors support parallel products, and, hence, a single CVE-referenced vulnerability typically affects multiple products (see Fig. 5). The effect is pronounced in the case of Microsoft

Table 2. The dataset

Windows				openSUSE				Ubuntu			
Prod	N	N_y	N_x	Prod	N	N_y	N_x	Prod	N	N_y	N_x
XP SP1	82	21	61	11.0	215	181	34	4.10	435	285	150
XP SP2	430	76	354	11.1	470	401	69	5.04	454	385	69
XP SP3	866	82	784	11.2	901	770	131	5.10	448	378	70
XP Prof. x64	260	53	207	11.3	1156	969	187	6.06 LTS	1476	1213	263
XP Prof. x64 SP2	922	89	833	11.4	670	510	160	6.10	473	398	75
Vista	276	36	240	12.3	1277	1143	134	7.04	468	394	74
Vista SP1	413	48	365	13.1	1242	1134	108	7.10	544	459	85
Vista SP2	1161	76	1085	13.2	557	498	59	8.04 LTS	1606	1186	420
Vista x64	276	36	240					8.10	710	580	130
Vista x64 SP1	416	48	368					9.04	667	503	164
Vista x64 SP2	1163	76	1087					9.10	746	553	193
7 i386	493	46	447					10.04 LTS	2373	1588	785
7 i386 SP1	953	47	906					10.10	883	589	294
7 x64	497	47	450					11.04	895	563	332
7 x64 SP1	966	48	918					11.10	1017	640	377
8 i386	702	19	683					12.04 LTS	2124	1321	803
8 x64	700	19	681					12.10	1029	652	377
								13.04	431	268	163
								13.10	464	329	135
								14.10	515	323	192
								14.04 LTS	938	558	380

Given the data collection in July 27, 2015, the column N reports the raw number of vulnerabilities that have affected a given product. The subsequent two symbols denote the number of vulnerabilities in the subsets of positive and non-positive values, respectively. The colored entries mark products that were still eligible for security patches at the time of data collection.

for which product variety has generally been larger within the observed product families. Moreover, there are (b) one-to-many references between advisories and CVEs, which leads to a notable operationalization problem. This issue is solved by using the largest per-product advisory timestamp (the latest day) for each referenced CVE. While also the reverse (the earliest dates) have been used [26], the present solution can be justified by maintaining that a given vulnerability was not entirely fixed and communicated to users until the last advisory released.

3.2 Control Variables

The same data sources are used for two control variables. The first, say S, denotes the *severity* of a vulnerability. The variable is based on the so-called base score in the Common Vulnerability Scoring System (CVSS). The scores range from zero to ten; the higher the value, the more severe the given vulnerability. The base

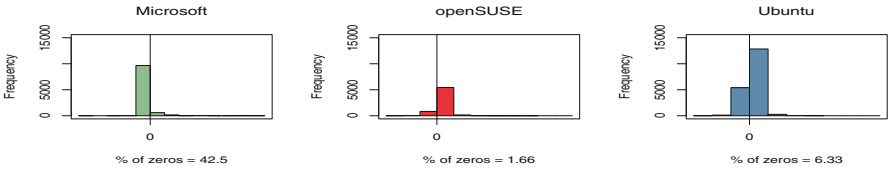

Fig. 3. Distribution of z_i

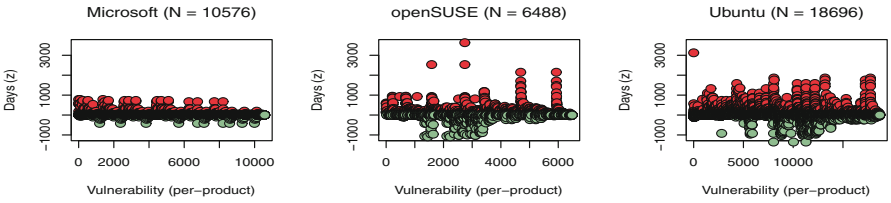

Fig. 4. Dispersion of z_i

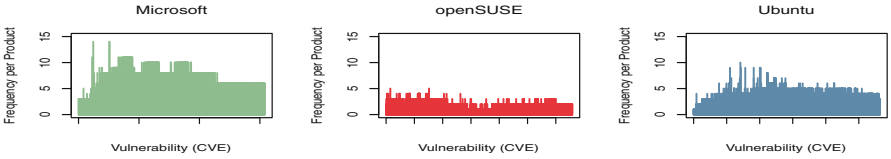

Fig. 5. Vulnerabilities per product

score is a composite metric that is computed from two groups of metrics that account for the impact and exploitability aspects of vulnerabilities [1]. Existing research has used both the base scores [3] and the more fine-grained individual metrics [26]. The theoretical rationale is simple either way: it is expected that the time lines are shorter for more severe vulnerabilities. The empirical usefulness of all CVSS metrics has been questioned [1], however, and, consequently, S is included only as a control variable without further contemplations.

The other control variable, say *references*, R, is operationalized as the cumulative amount of per-release security advisory references that were made to the same CVE identifier. A comparable operationalization has been used previously to measure the quality of security patches [26]. As the construct validity of R as a measure of patch (or patching) quality is arguably rather limited, also this quantity enters as a statistical control variable. In the ideal case of simplicity, both S and R can be eliminated as empirically redundant.

3.3 Methods

The linear equation in (2) is estimated as it reads – as a regression model:

$$\mathcal{M}_1: \; f(u_i) = \beta_0 + \beta_1 f(A_i) + \beta_2 f(A_i)^2 + \beta_3 f(S_i) + \beta_4 f(R_i) + \epsilon_i, \qquad (4)$$

where u_i refers either to y_i or $|x_i|$, $f(v) = \ln(v + 1)$, and ϵ_t is a residual term. The quadratic age term, $f(A_i)^2$, is included to account potential curvature. The function$f(v)$ is used to improve the assumption of normality [3]. Three smaller parsimonious models are sought with restrictions

$$\mathcal{M}_2 : \ \beta_2 = 0, \ \ \mathcal{M}_3 : \ \beta_1 = \beta_2 = 0, \quad \text{and} \quad \mathcal{M}_4 : \ \beta_1 = \beta_2 = \beta_4 = 0. \tag{5}$$

The model $\mathcal{M}_3$ assesses the actual age assertion by excluding both $f(A_i)$ and $f(A_i)^2$. In other words, the assertion implies that $\beta_1 \neq \beta_2 \neq 0$. While the single restriction $\beta_2 = 0$ can be evaluated with a t-test, all can be evaluated with the so-called Wald-tests [10], including the joint restrictions over the parameters. All models were estimated also with a so-called fixed effects strategy by including a set of dummy variables to control for the per-product heterogeneity. As none of the estimates changed notably, the results reported omit these effects, however.

Besides evaluating the assumption of normality with Q-Q plots, three diagnostic checks are computed to assess the overall statistical fits: (a) the so-called RESET test is used to account for potential non-linearity; (b) residual (auto)correlation (dependence between u_i and u_{i+1}) is evaluated with the Breusch-Godfrey test; and (c) heteroskedasticity (systematic dispersion in the residual terms) is checked with the Breusch-Pagan test (for details see, e.g., [10]). The rationale for the checks (b) and (c) relates to the sampling strategy that allows per-product "duplicate" vulnerabilities to enter into the estimation samples. Finally, estimation is carried out with R, using the *lmtest* package [29] for the three statistical tests. If required, the estimates are adjusted by using suitable covariance matrix estimators from the *sandwich* package [28]. The general performance is evaluated with adjusted R^2_a-values; higher values are better.

3.4 Sampling

The three vendor subsamples are analyzed as three separate, statistically independent cases. In general, however, the data in Table 2 cannot be directly estimated because the software life cycles vary across the observed products. To account for this variance, a vendor-specific sampling strategy is used on per-product basis in two steps. Given a vendor's all observed products, random samples (without replacement) are first picked for the vendor's each product. In the second step these random samples are merged into a per-vendor estimation sample, after which the sample split into y_i and $|x_i|$ observations is computed according to the $z_i > 0$ and $z_i \leq 0$ criteria, respectively.

This procedure ensures that estimation is balanced with respect to a vendor's varying per-release software life cycle lengths. The size of these per-product random samples is defined as the minimum per-product sample size in Table 2. For instance, 431 random vulnerabilities are sampled for each of the Ubuntu releases, given the minimum sample size of 431 observed vulnerabilities in Ubuntu 13.04. Since 431 random vulnerabilities are sampled also for 12.04 LTS, the estimation is not biased towards the longer age and, consequently, the larger amount of vulnerabilities in the long-term support releases. This random sampling procedure

is repeated 100 times. The statistical results are recomputed in each iteration. Because the standard deviations are rather small across the iterations, arithmetic mean is used for summarizing the results.

3.5 Results

The regression analysis can be started from the diagnostic checks. To begin with, normal approximation is generally reasonable with the $f(v)$ transformation. By embedding normal Q-Q curves for the residuals from the two hundred regression models, the normality observation is illustrated in Fig. 6 for openSUSE. Thus, inference with statistical significance is generally justified. The other diagnostic checks indicate some problems, however. Only Microsoft passes all of the three formal tests; the models for openSUSE and Ubuntu indicate heteroskedasticity in the residual terms, and there appears to be also some (auto)correlation in the models for Ubuntu. To account for these issues, the results are recomputed with the Newey-West covariance matrix estimator [10,17] for openSUSE and Ubuntu.

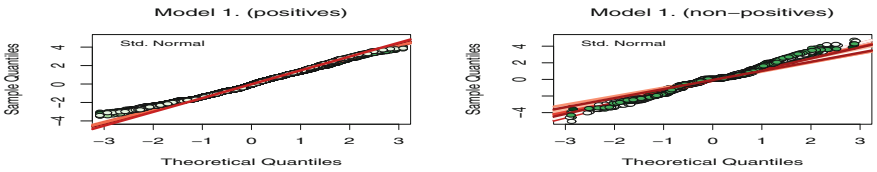

Fig. 6. Normal Q-Q plots for openSUSE ($\mathcal{M}_1$)

The regression results can be packed into the tight summary shown in Table 3. The panel (a) shows the mean p-values from the Wald-tests for both subsets. The conclusion is clear for Microsoft: the full model in (4) can be reduced all the way to $\mathcal{M}_4$ in (5). That is, the age assertion does not hold for Microsoft. The reduction, $\mathcal{M}_1 \mapsto \mathcal{M}_3$, cannot be done for Ubuntu and openSUSE with respect to the subsets of positive z_i values. As expected, the coefficients are positive for the $f(A_i)$ term in these subsets, as seen from the second column in the panel (b). However, the same does not hold in the $z \leq 0$ subsets; the reduction to $\mathcal{M}_3$ is applicable to all vendors according to the fourth column in the panel (a). Because the age assertion was specified in a dual fashion (see Sect. 2), openSUSE and Ubuntu pass the assertion only partially. It is also worth remarking that the coefficients for the severity variable attain their expected negative signs; severe vulnerabilities are processed slightly faster. Finally, the statistical performance is very limited; only approximately 10 % of the total variance is explained at best. Taken together, these observations suggest the age assertion cannot be used to characterize the contemporary operating system population.

Table 3. Regression results[1]

	(a) Wald-tests (mean p-values)						(b) Estimates (means, $\beta_2 = 0$)				
	$\mathcal{M}_1 \mapsto \mathcal{M}_2$		$\mathcal{M}_1 \mapsto \mathcal{M}_3$		$\mathcal{M}_1 \mapsto \mathcal{M}_4$		$\widehat{\beta}_0$	$\widehat{\beta}_1$	$\widehat{\beta}_3$	$\widehat{\beta}_4$	R^2_a
Subset	y	x	y	x	y	x	y	y	y	y	y
Microsoft	0.52	0.48	0.49	0.14	0.07	0.58	6.00	-0.06	-1.55	2.47	0.10
openSUSE[2]	0.30	0.36	0.00	0.24	0.00	0.00	3.00	0.18	-0.76	1.04	0.08
Ubuntu[2]	0.57	0.08	0.00	0.20	0.10	0.00	3.31	0.17	-0.61	0.25	0.04

[1] Colored entries refer to $p < 0.05$; due to lack of space, values 0.00 denote $p < 0.01$.

[2] Results are computed with the Newey-West estimator.

4 Discussion

This empirical paper investigated an assertion that the age of software products can be used to predict the time delays between security advisory releases and the publication of CVEs. The paper finds no systematic empirical evidence for such an assertion. Three conclusions can be drawn from the regression analysis of nearly fifty operating system releases. First, there is no notable statistical relationship between the age of Microsoft Windows releases and the turnaround times between Microsoft, Inc. and the institutional setup represented by MITRE, NVD, and associated parties. Second, only the handling of vulnerabilities that take the route (b) in Fig. 2 is mildly correlated with the age of openSUSE and Ubuntu releases, but otherwise the results are mixed. Last, all of the examined variables – age, severity, and amount of CVE-references – seem to be more or less irrelevant for predicting the turnaround times. Only about a ten percent of the total variation is explained by the three variables.

Thus, the observations constitute a negative result with respect to the theorized age assertion. By implication, it seems sensible to recommend that no particular attention is required in release engineering strategies for the concern that long support periods would systematically lengthen the delays between the patching and publication VLC states. In this sense the negative result is a positive result: there is no particular reason for users to fear that patching would take long for an old operating system release.

Although the results are generally in accordance with previous observations regarding the severity of vulnerabilities [26], the overall lack of explanatory power indicates that neither the statistical severity scoring deserves particular mentions in the strategies. Rather, something else largely explains the variation in the turnaround times. A plausible technical explanation may relate to the large amount of code that is shared particularly between successive operating system releases. That is, patching a vulnerability in a new release may often involve the same code that is present in an older release, which implies that the turnaround times should be rather similar for both releases.

Because the results are particularly clear for the observed Microsoft Windows releases, it seems reasonable to end with a remark about the much more

fundamental assertion placed over the closed and open source continuum [23,24]. Because Microsoft is generally fast at the coordination (and irrespectively of the three variables examined in this paper), a particularly worthwhile topic for further research is to map and examine the potential reasons that may hinder the efficiency at the open source front. As has been suspected [13], narrow time lines are explicitly targeted by Microsoft, but it is unclear why large open source vendors are unable to achieve the same level of efficiency. The reasons for this divergence are presumably economical and socio-technical rather than technical.

References

1. Allodi, L., Massacci, F.: Comparing vulnerability severity and exploits using case-control studies. ACM Trans. Inf. Syst. Secur. **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. Inf. Econ. Policy **22**(2), 164–177 (2010)
4. Canonical Ltd.: Releases (2015). https://wiki.ubuntu.com/Releases. July 2015
5. Canonical Ltd.: Ubuntu Security Notices (2015). http://www.ubuntu.com/usn/. March 2015
6. Cavusoglu, H., Cavusoglu, H., Raghunathan, R.: Efficiency of vulnerability disclosure mechanisms to disseminate vulnerability knowledge. IEEE Trans. Softw. Eng. **33**(3), 171–185 (2007)
7. 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)
8. 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)
9. Khomh, F., Adams, B., Dhaliwal, T., Zou, Y.: Understanding the impact of rapid releases on software quality: the case of Firefox. Empir. Softw. Eng. **20**(2), 336–373 (2015)
10. Kleiber, C., Zeileis, A.: Applied Econometrics with R. Springer, Berlin (2010)
11. Li, P., Rao, R.: An examination of private intermediaries' roles in software vulnerability disclosure. Inf. Syst. Front. **9**(5), 531–539 (2007)
12. Mäntylä, M.V., Adams, B., Khomh, F., Engström, E., Petersen, K.: On rapid releases and software testing: a case study and a semi-systematic literature review. Empir. Softw. Eng. **20**(5), 1384–1425 (2014)
13. 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)
14. 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)

15. Microsoft Inc.: Microsoft Security Bulletin Data (2015). http://www.microsoft.com/en-us/download/details.aspx?id=36982. July 2015
16. Microsoft Inc.: Windows Life Cycle Fact Sheet (2015). http://windows.microsoft.com/en-us/windows/lifecycle. July 2015
17. Newey, W.K., West, K.D.: A simple, positive-definite, heteroskedasticity and autocorrelation consistent covariance matrix. Econometrica **55**(3), 703–708 (1987)
18. 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)
19. 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). https://nvd.nist.gov/download.cfm. June 2015
20. Novell Inc. and others.: openSUSE: Lifetime (2015). https://en.opensuse.org/Lifetime. July 2015
21. Novell Inc. and others: openSUSE: Roadmap (2015). https://en.opensuse.org/openSUSE:Roadmap. July 2015
22. Ruohonen, J., Hyrynsalmi, S., Leppänen, V.: The sigmoidal growth of operating system security vulnerabilities: an empirical revisit. Comput. Secur. **55**, 1–20 (2015)
23. Schryen, G.: Is open source security a Myth? Commun. ACM **54**(5), 130–140 (2011)
24. 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)
25. SUSE LLC: Published SUSE Linux Security Updates by CVE Number (2015). https://www.suse.com/security/cve/. June 2015
26. Temizkan, O., Kumar, R.L., Park, S., Subramaniam, C.: Patch release behaviors of software vendors in response to vulnerabilities: an empirical analysis. J. Manag. Inf. Syst. **28**(4), 305–337 (2012)
27. 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)
28. Zeileis, A.: Econometric computing with HC and HAC covariance matrix estimators. J. Stat. Softw. **11**(10), 1–17 (2004)
29. Zeileis, A., Hothorn, T.: Diagnostic checking in regression relationships. R News **2**(3), 7–10 (2002)