Int. J. Inf. Secur. (2017) 16:673–690
DOI 10.1007/s10207-016-0345-x

REGULAR CONTRIBUTION

Periodicity in software vulnerability discovery, patching and exploitation

HyunChul Joh[1] · **Yashwant K. Malaiya**[2]

Published online: 19 July 2016

Abstract Periodicity in key processes related to software vulnerabilities need to be taken into account for assessing security at a given time. Here, we examine the actual multi-year field datasets for some of the most used software systems (operating systems and Web-related software) for potential annual variations in vulnerability discovery processes. We also examine weekly periodicity in the patching and exploitation of the vulnerabilities. Accurate projections of the vulnerability discovery process are required to optimally allocate the effort needed to develop patches for handling discovered vulnerabilities. A time series analysis that combines the periodic pattern and longer-term trends allows the developers to predict future needs more accurately. We analyze eighteen datasets of software systems for annual seasonality in their vulnerability discovery processes. This analysis shows that there are indeed repetitive annual patterns. Next, some of the datasets from a large number of major organizations that record the result of daily scans are examined for potential weekly periodicity and its statistical significance. The results show a 7-day periodicity in the presence of unpatched vulnerabilities, as well as in the exploitation pattern. The seasonal index approach is used to examine the statistical significance of the observed periodicity. The autocorrelation function is used to identify the exact periodicity. The results show that periodicity needs to be considered for optimal resource allocations and for evaluation of security risks.

Keywords Vulnerability · Laws of vulnerabilities · Seasonality · Periodicity · Operating system

1 Introduction

As software systems become larger and more complex, the number of defects they contain has become a major consideration. A fraction of all defects are security related and are termed vulnerabilities. Vulnerabilities are major concerns in all systems whether they are open source or commercial. A famous quotation is attributed to Lord Kelvin: "If you cannot measure it, you cannot improve it." Both developers and users need to be able to assess the security level of software systems in order to improve them. The security of a software system is impacted by the presence of unpatched vulnerabilities which are discovered from time to time and can be potentially exploited.

A number of studies related to the evaluation of computer security have been carried out. However, most of them have been qualitative. Quantitative methods are well established in fields such as performance assessment, metric measurement, functional evaluation, or statistical modeling, but these are relatively new in the area of security evaluation. Quantitative statistical analysis methods permit developers to estimate future trends much more accurately because they provide actual data-driven estimation approaches [6]. Quantitative methods will make it possible for end-users to objectively assess the risk posed by vulnerabilities in software systems and their potential breaches. Researchers have begun to explore some of the quantitative characteristics of vulnerabilities as the required datasets are becoming sufficiently

✉ HyunChul Joh
joh@kiu.kr

Yashwant K. Malaiya
malaiya@cs.colostate.edu

[1] Department of Computer Engineering, Kyungil University, Gyeongsan, Korea

[2] Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA

large to be analyzed [20,31]. Nevertheless, quantitative risk analysis in information security systems is still in its infancy.

Security vulnerabilities in major applications that have been discovered and disclosed but not remedied represent great risk of both organizations and individuals. They include major classes of software systems such as operating systems (OSes), Web servers and browsers. The OSes form the complex foundation for all computing systems, while Web servers and browsers provide the connectivity among modern computing systems due to the Internet. Unfortunately, each year, a large number of vulnerabilities are detected in OSes and Web-related software systems that represent a major security risk [33]. If we could predict the expected vulnerability discovery pattern and the attributes of the vulnerabilities discovered, the needed resources could be allocated at the right time for corrective measures, which would greatly reduce security risks. Such methods could also be utilized by end-users to assess risks and be prepared to take remedial action when potential security breaches occur.

Figure 1 represents the box plots for the number of newly discovered vulnerabilities in the three types of software systems grouped for each month. Table 1 shows the number of known vulnerabilities for each software system, together with the examined periods. Here, software systems are categorized into Windows OSes (Windows NT, Windows 95, Windows 98, Windows 2000, Windows XP, and Windows 7), non-Windows OSes (iPhone OS, MAC OSX, Red Hat Linux Enterprise, AIX, Android, and ChromeOS), and Web-related software systems [Apache http server, IIS, Internet Explorer, Firefox, Safari, and Java (JRE)]. Figure 2 shows the number of vulnerabilities found each calendar month. All datasets were extracted from the National Vulnerability Database (NVD; http://nvd.nist.gov) on August 2014. In Table 1, periods are determined by the time when the first and the last vulnerabilities were reported in each software system in the NVD datasets.

In Fig. 1, it is observed visually that all the three software groups peak in certain months. For the Windows OSes, a strong year-end peak and a somewhat weaker mid-year peak are observed. On the other hand, the Web-related software systems have relatively fewer vulnerabilities reported year-end, but a strong mid-year peak. The non-Windows OSes have strong peaks between mid-year and year-end, March and September. We will analyze these patterns later in this paper using statistical methods to see whether the differences are statistically significant. A consistent variation in annual box plots suggests a seasonal pattern that should be taken into account in making more accurate predictions about

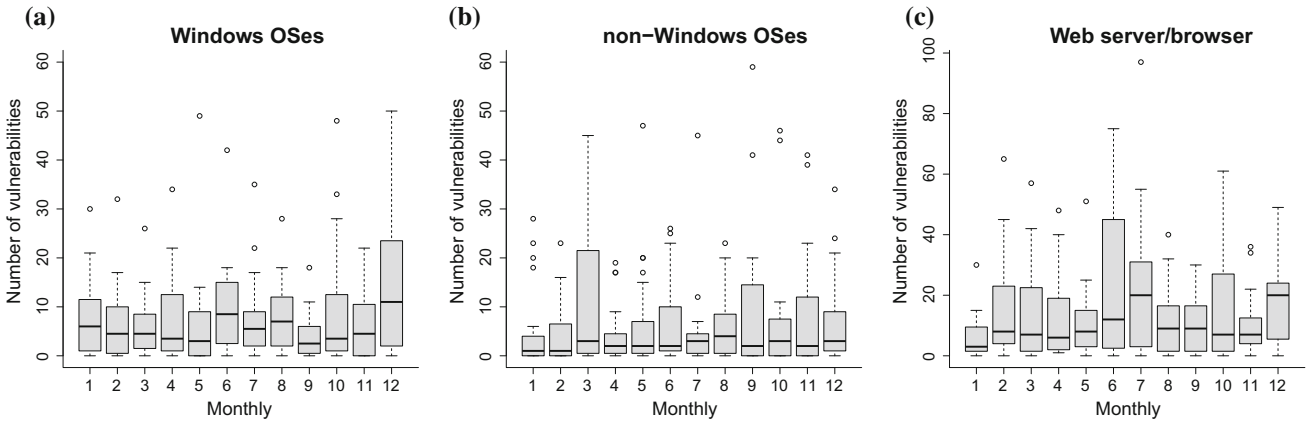

Fig. 1 Box plots for each software group's cumulative number of vulnerabilities in month. **a** Windows OSes. **b** non-Windows OSes. **c** Web-related software

Table 1 Number of vulnerabilities for the eighteen software systems and observed period

Win:	WinNT	Win95	Win98	Win2K	WinXP	Win7
Vul. #	470	47	90	505	724	342
Period	1995–2010	1997–2009	1999–2009	1997–2012	2000–2014	2009–2014
non-Win:	iPhoneOS	OSX	RHEL	AIX	Android	ChromeOS
Vul. #	359	899	274	314	37	47
Period	2007–2014	1997–2014	1996–2014	1992–2014	2009–2014	2010–2014
Web:	Apache	IIS	IE	Firefox	Safari	Java (JRE)
Vul. #	183	151	967	1088	510	434
Period	1996–2014	1996–2014	1997–2014	2003–2014	2003–2014	2001–2013

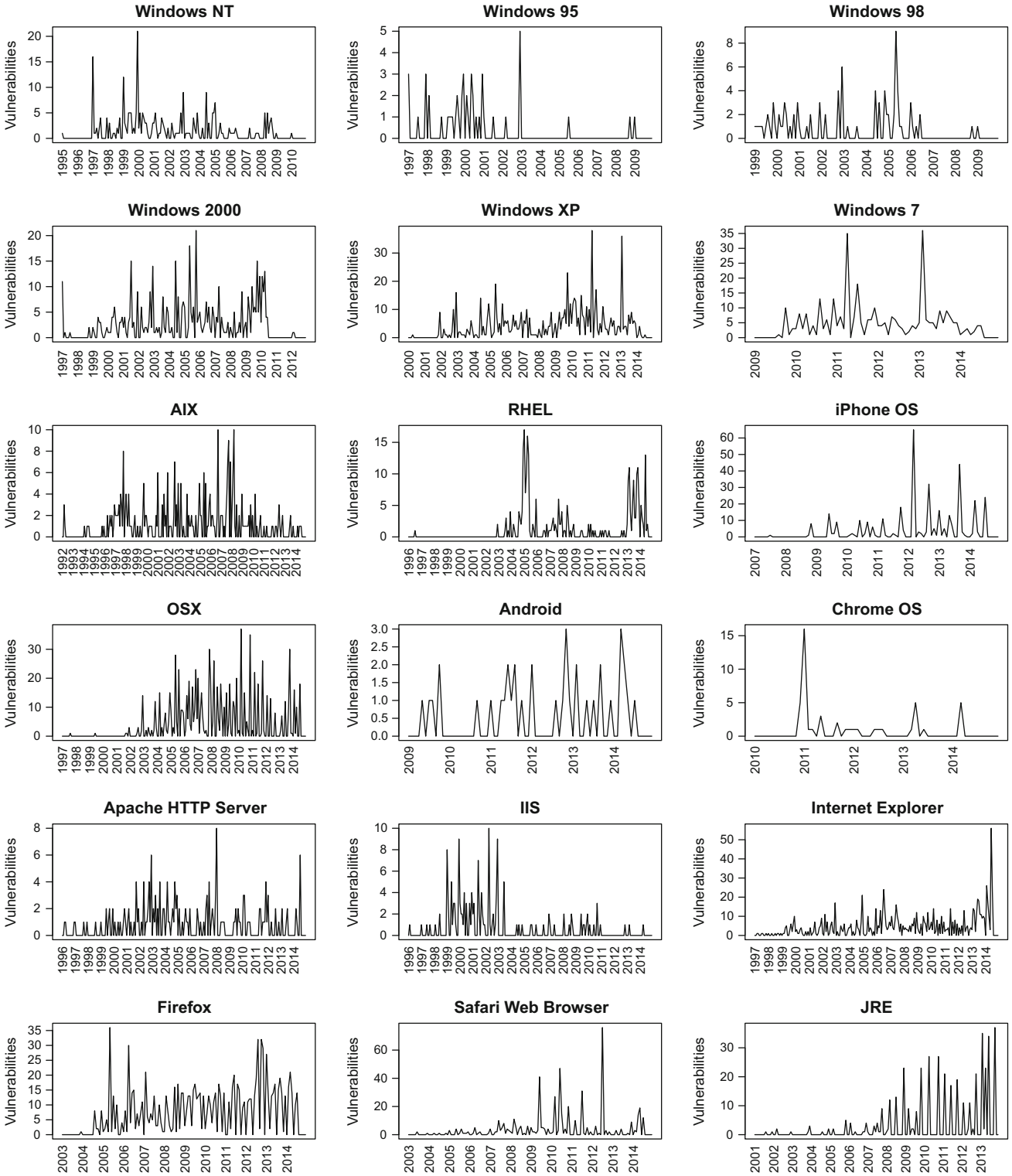

Fig. 2 Run chart for the number of monthly vulnerabilities, with calendar time

the number of vulnerabilities expected to be discovered in a future period. We examine the question of the existence of annual and weekly patterns and their significance for activities (discovery, patching and exploitation) related to software vulnerabilities using actual datasets in a quantitative manner [15,16].

The paper is organized as follows. Section 2 discusses seasonality-related research in other fields. Section 3 presents

the background research related to security vulnerabilities. Section 4 concisely describes the statistical methods used in this paper. The next section presents an analysis of potential seasonality in the eighteen software systems. In Sect. 6, we will examine the 7-day periodic behavior in the presence of unpatched vulnerabilities, as well as the exploitation pattern. Section 7 discusses possible factors affecting the periodic behavior in the vulnerability-related activities. Conclusions will be given in Sect. 8.

2 Related works on seasonality

Periodic behavior such as seasonal or weekend effects are well-established research areas in other disciplines such as the stock market [12], high-performance computing systems [34], epidemiology [30], power transmission [32], marine biology [21], and birth defects [7]. This section reviews them briefly.

Stocks tend to have relatively higher returns for certain specific calendar months. The higher return between November and April is termed the Halloween Effect (http://ssrn.com/abstract=901088). Heston and Sadka [12] have identified a repetitive and distinctive pattern with lags of 12, 24 and 36 months in the data of [14] caused by the persistent seasonal effect in stock returns. The results could potentially be useful for developing seasonal stock market strategies. Tran and Reed [34] have examined the software application performance, using the fact that application I/O patterns are bursty in scientific codes due either to periodic checkpoints or nested loops, and such bursty patterns could cause an overflow in system resource usage. The authors have tried to predict the resulting I/O request patterns using time series models. This kind of access pattern forecast could be used to make pre-fetch decisions during application execution time.

In epidemiology, Rios et al. [30] attempted to determine whether pulmonary tuberculosis has an annual seasonal pattern by using the autoregressive integrated moving average (ARIMA) time series model using autocorrelation function (ACF) and partial ACF. The seasonal trend for the disease could be caused by a rise in indoor activities in winter, thereby increasing the risk of exposure of healthy persons to bacilli from other infected persons. Another reason could be the fact that infections of viral etiology are more frequent in winter causing immunological deficiency. The model developed by the authors can also determine whether the incidence of the disease is greater than that forecast by the model so that the model could be used to assess the quality of the existing preventative measures.

For power transmission in utilities, Salehian [32] has attempted to model thermal rating patterns that are influenced by weather, using the ARIMA time series model for forecasting. Forecasting the thermal capabilities of the line would allow meeting contractual power delivery obligations. In marine biology, Maes et al. [21] have tried to determine how elements affect the abundance of fish, which are exposed to great environmental variabilities in the form of dissolved oxygen, temperature, water quality, salinity, prey, etc. The results show that the life cycles of marine organisms have clear seasonal patterns in growth, reproduction, and abundance.

Carrion-Baralt et al. [7] have investigated whether the birth of a schizophrenic infant, which occurs more frequently during winter, is actually due to severe winter temperatures, as was thought previously. In contrast to previous studies, the authors conducted their work on a tropical island having no severe winter weather and drew the conclusion that extreme temperatures are not a sufficient explanation for this phenomenon. Lastly, in [35], the authors tried to find any seasonal patterns of tobacco smoking behavior using online search information, Google Trends. They first collected the search query of "smoking" from the search engine in six nations. Then, periodogram method was applied to determine whether the seasonality of smoking exists in the data. They also calculated the pairwise cross-correlations among the six datasets. The result shows that the similar weather condition on the same hemisphere leads to the similar smoking-related search behavior. Since seasonal periodicity analysis is a well-known statistical approach in the repertoire of many researchers and analysts in several disciplines as discussed above, some well-developed analytical techniques exist that can be applied to the information security field.

3 Recent research in security vulnerabilities

A software vulnerability can be defined as a defect or weakness in the security system which might be exploited by a malicious user causing loss or harm [26]. Thus far, only limited work has been done to characterize security vulnerabilities in a quantitative manner. There are a few standards commonly used by researchers to render the vulnerabilities found measurable. These include Common Vulnerability Exposures (CVE; http://cve.mitre.org), Common Weakness Enumeration (CWE; http://cwe.mitre.org), and Common Vulnerability Scoring System (CVSS; http://www.first.org).

CVE is a publicly available, free-for-use list or dictionary of standardized identifiers concerning common computer vulnerabilities and exposures. CVE's common identifiers enable data exchange among security products and provide a baseline index point for evaluating coverage of tools and services.

CWE is a list of software weakness types that is intended to be a complete dictionary of software weaknesses. A unique number is assigned to each weakness type which allows a clear description, and hence better management of software

weaknesses related to architecture and design. CWE provides a very detailed classification using programming, design, and architectural errors that led to vulnerabilities.

CVSS, the Common Vulnerability Scoring System, has been adopted by many vendors since its launch in 2004. This system, whose initial design was finalized in 2007, is now in its second version. CVSS includes three metric groups, which are termed base, temporal, and environmental. While the base metric group is required for the final CVSS score, the other two groups are optional. The score is in the range of [0.0, 10.0]. Scores close to 0.0 are relatively benign vulnerabilities, whereas scores close to 10.0 mean that the vulnerabilities are more likely to encounter exploitation causing serious consequences. On May 2012, the CVSS Special Interest Group started to develop the third version, and on June 2014, preview of version 3 has been available.

Some vulnerability discovery models (VDMs) that model the vulnerability discovery process have been proposed recently. These include Andersons thermodynamic model [3], Alhazmi-Malaiya Logistic (AML) model [1], and Rescorlas linear/exponential models [28]. Some classical software reliability growth models have also been used as [25].

Recently, the impact of evolution caused by successive software versions on the vulnerability discovery process have been examined [8,18]. In their study, Kim et al. [18] have considered the impact of sharing code across successive versions. They have proposed an enhanced version of the AML model by taking into consideration the superposition [10] of the discovery processes for multiple version software systems and verified their model by examining two open source software systems. Chen et al. [8] have presented a multi-cycle vulnerability discovery model incorporating a sinusoidal behavior. The multi-cycle model attempts to model the relationship between the number of vulnerabilities and their release time and is compared with other VDMs using datasets from eight Windows OSes.

Ozment [24] has proposed a standard set of terms for measuring characteristics of vulnerabilities and their discovery processes. Alhazmi and Malaiya [1] have compared several VDMs by fitting the data for major operating systems and have shown that AML fits better than other models in most cases. However, since AML is a symmetrical model, it may not perform well with asymmetric discovery patterns. Joh and Malaiya [17] have examined some S-shaped distributions which can model an asymmetrical behavior. Condon et al. [9] have used software security incident datasets to compare forecasts using time series models and using software reliability growth models. There are a large number of vulnerability databases and security advisories on the Web. As a result, selecting the appropriate data source for quantitative analysis is not an easy task. Massacci and Nguyen [22] have examined the accuracy of sampling by the researchers for their quantitative vulnerability analyses. Many vulnerability databases provide de facto standard information such as CVE identifier numbers and CVSS. The best known vulnerability databases include NVD, Open Source Vulnerability Database (OSVDB) (http://osvdb.org), US-CERT (http://www.kb.cert.org/vuls), IBM X-Force (http://xforce.iss.net) and Secunia (http://secunia.com).

In this paper, we have used the NVD database for long-term annual periodicity analysis, because it provides the most extensive datasets, and the data is collected and organized by using specific standards. Since the NVD project is sponsored by the US Department of Homeland Security and maintained by National Institute of Standard and Technology, it can be considered a standard source. Moreover, NVD uses CVE identifiers, so that any updates to CVE immediately appear on NVD. Also, in Sect. 6, which addresses shorter term periodicity, an extensive amount of data gathered globally by Qualys [27] using automated scans, such as exploitation patterns and the presence of unpatched vulnerabilities, is used to examine potential weekly periodicity in vulnerability-related activities.

4 Statistical methods in seasonality analysis

The seasonal index is the most common measure used in seasonality analysis. A seasonal index states to what extent the average for a particular period tends to be above (or below) the expected value. The monthly seasonal index values are given by

$$s_i = \frac{d_i}{d} \tag{1}$$

where, s_i is the seasonal index for the ith month, d_i is the mean value of the ith month, and d is a grand average (http://home.ubalt.edu/ntsbarsh/business-stat/stat-data/forecast.htm). Thus, for example, a monthly seasonal index of 1.25 indicates that the expected value for that month is 25 greater than 1/12 of the overall annual average where the expected value is 1. To determine whether the seasonal indices are statistically significant, the Chi-square test is also applied. To evaluate the significance of the non-uniformity of the distribution in calculated indices, we tested the grand total of each month against the expected value (total vulnerabilities divided by 12). The Chi-square statistic is calculated as

$$\chi_s^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)^2}{e_i} \tag{2}$$

where o_i and e_i are the observed and expected values at the ith time point, respectively. For the test to be acceptable, the Chi-square statistic value (χ_s^2) should be less than

the corresponding Chi-square critical value (χ_c^2) with the given alpha level and degrees of freedom. The p value produced by the test represents the probability that a value of the statistic, at least as high as the value calculated in Eq. 2 could have occurred by chance. In this paper, to confirm the non-uniformity of the distribution in calculated indices, p values need to be smaller than 0.05 since we are using the alpha level of 0.05. Goonatilake et al. [11] describe how to apply Chi-square test in software security. To pinpoint which months seasonal index is statistically greater or less than others, analysis of variance (ANOVA) with Fisher's least significant difference (LSD) tests [23] is conducted on the calculated seasonal indices. When a result from the ANOVA test is significant, it indicates there is at least one group that differs from the other groups. However, an ANOVA analysis can only determine whether the indices among the 12 months are statistically the same or not, it cannot determine which months index is higher or lower than others.

LSD is a test for comparing average values from treatment groups after the ANOVA null hypothesis of equal means among the groups has been rejected. Once the ANOVA null hypothesis is rejected, researchers need to check pairwise comparisons of means from each group.

When there are twelve seasonal indices for each of the months and μ_i is the mean seasonal index value for month i, then LSD is used to test the null hypothesis that $\mu_i = \mu_j$ where $1 \leq \{i \text{ and } j\} \leq 12$, i and j are integer, and $i \neq j$. When the calculated $\text{LSD}_{i,j}$ is $|\mu_i - \mu_j| \geq \text{LSD}_{i,j}$, then the null hypothesis of $\mu_i = \mu_j$ will be rejected, confirming that there is a statistically significant difference between the two groups. $\text{LSD}_{i,j}$ can be obtained by [23]:

$$\text{LSD}_{i,j} = t_{\alpha/2,df}\sqrt{\text{MS}\left(\frac{1}{n_i}+\frac{1}{n_j}\right)} \tag{3}$$

where df is degrees of freedom, MS is mean square value from the ANOVA test, and n_i is number of observations in group i. $t_{\alpha/2,df}$ value can be obtained from the Student's t-distribution table.

The other approach to characterize periodicity is to use the autocorrelation function (ACF). ACF analysis gives us specific relationship information in terms of the related time units such as month or day. With time series values of $z_b, z_{b+1}, \ldots, z_n$, the ACF at time lag k, denoted by r_k , is [5]:

$$r_k = \frac{\sum_{t=b}^{n-k}(z_t-\bar{z})(z_{t+k}-\bar{z})}{\sum_{t=b}^{n}(z_t-\bar{z})^2}, \quad \text{where } \bar{z} = \frac{\sum_{t=b}^{n} z_t}{n-b+1} \tag{4}$$

ACF measures the linear relationship between time series observations separated by a lag of k time units. When an ACF value is located outside chosen upper or lower confidence intervals, there is a statistically significant periodic relationship associated with that time lag. An event occurring at time $t+k(k>0)$ is said to lag behind an event occurring at time t, the extent of the lag being k. In this paper, the seasonal index analysis with the Chi-square test, ANOVA test, LSD test, and the ACF analysis is used to establish the annual and weekly periodic behaviors discussed below.

5 Annual seasonal variations in vulnerability discovery processes

Figure 2 gives plots for the number of vulnerabilities disclosed each month for the eighteen software products. It can be seen that, for each software group, certain months tend to have more vulnerabilities then others. For example, in Windows group, year-end has peak values, while mid-year (summer) peaks are shown in the Web-related software group, suggesting the possibility of seasonality. In this section, we examine the significance of this observed annual seasonality. We group the software systems into three categories for convenience—Windows OSes, non-Windows OSes, and Web-related systems. We will examine the null hypothesis H_0 that the seasonal indices for the 12 months are not all significantly different from each other. The same methods will be used in the next section for checking weekly periodic behaviors in the exploitation pattern and prevalence of unpatched vulnerabilities.

5.1 Seasonal index analysis

A time series data is not uniformly distributed, and periodic patterns are present in a dataset when certain months have significantly more incidents of reported vulnerabilities than other months. Table 2 shows seasonal indices for the 12 months from each system, as described in Sect. 4 above. In Fig. 3, for Windows OSes, seasonal index values for mid-year (summer) and year-end (winter) tend to have higher values than index value of 1.0, which represents the expected value. Consequently, the months between the two peaks tend to have the expected seasonal index value less than 1.0. The values for March and September are lower than others.

For the non-Windows OSes, there is no clear consistent mid-year seasonal patterns, but March, September, and December seem to have higher values. In contrast, clear consistent mid-year seasonal patterns are found among the indices from Web-related software systems. As for the Web-related software systems, IIS and IE datasets display a pattern similar to that of Windows OSes, which are the native platforms for the two Web-related systems; the mid-year and year-end periods tend to have more vulnerabilities than other times for the four systems. The mid-year peak may explain the higher third quarter advisories for Microsoft products

Table 2 Vulnerability discovery process seasonal indices

	WinNT	Win95	Win98	Win2K	WinXP	Win7
JAN	1.9483	1.2766	1.0667	0.8792	0.5967	0.5965
FEB	0.8413	1.2766	0.5333	0.6653	1.2928	2.0000
MAR	0.5756	0.7660	0.6667	0.8317	0.7293	0.7018
APR	0.8413	0.2553	0.6667	1.0455	1.4751	1.7544
MAY	0.7528	1.0213	1.8667	0.9505	0.6630	0.5263
JUN	1.2841	0.5106	1.6000	1.4495	1.1602	0.8421
JUL	0.8413	1.0213	0.5333	0.9505	0.7956	1.2632
AUG	0.7970	0.7660	1.0667	1.0218	1.0276	0.9474
SEP	0.5756	0.2553	0.1333	0.4040	0.6298	0.4912
OCT	0.6642	0.7660	0.9333	1.4733	1.2928	1.0877
NOV	0.7970	0.5106	0.9333	0.8317	0.6796	0.5965
DEC	2.0812	3.5745	2.0000	1.4970	1.6575	1.1930
χ_c^2	19.6751	19.6751	19.6751	19.6751	19.6751	19.6751
χ_s^2	335.1587	80.4255	116.5333	556.3010	813.7127	416.3509
p value	3.33506E−65	1.22095E−12	9.01362E−20	3.0778E−112	2.1527E−167	2.06099E−82
	iPhoneOS	OSX	RHEL	AIX	Android	ChromeOS
JAN	0.5348	0.4405	0.9635	0.9172	0.9730	4.3404
FEB	0.1337	0.7341	1.0511	0.8408	0.6486	0.5106
MAR	3.5097	1.8020	1.3577	1.2994	0.9730	1.7872
APR	0.1671	0.4939	1.1825	0.6879	0.9730	1.2766
MAY	0.5348	1.2280	0.9635	0.7261	1.2973	0.7660
JUN	1.1031	1.0145	1.0949	0.7643	0.6486	0.5106
JUL	0.9694	0.4271	0.7007	0.9172	1.2973	0.2553
AUG	0.2674	1.0011	0.7007	1.1847	1.2973	0.2553
SEP	3.1421	0.7608	0.6131	1.3376	0.9730	0.5106
OCT	0.7688	1.1613	0.8759	1.0701	1.2973	0.0000
NOV	0.8022	1.8554	0.8321	0.7643	0.9730	0.2553
DEC	0.0669	1.0812	1.6642	1.4904	0.6486	1.5319
χ_c^2	19.6751	19.6751	19.6751	19.6751	19.6751	19.6751
χ_s^2	786.0836	1085.1657	296.5839	335.0828	39.2432	108.0000
p value	1.8417E−161	8.8941E−226	4.5928E−57	3.4605E−65	4.81608E−05	4.59238E−18
	Apache	IIS	IE	Firefox	Safari	JRE
JAN	0.9180	1.1921	0.4219	0.3860	0.3529	0.0553
FEB	0.7869	1.0331	1.0796	1.1029	0.1882	2.0184
MAR	1.0492	0.2384	0.9183	1.0147	1.4588	0.7465
APR	0.6557	0.9536	0.7446	1.1250	0.6824	1.3825
MAY	0.7869	0.8742	0.7322	0.9375	0.7529	0.2212
JUN	0.9180	1.5894	2.1096	1.1912	2.5647	1.7972
JUL	1.1803	0.9536	1.3030	1.3346	3.5765	0.4700
AUG	1.2459	0.6358	1.0176	1.0257	0.3765	0.3594
SEP	0.7869	0.9536	0.6205	1.1581	0.4941	0.0277
OCT	1.1803	0.5563	0.8687	0.9926	0.2118	3.1521
NOV	0.9180	0.7152	0.5460	0.7390	0.8941	0.8295

Table 2 continued

	Apache	IIS	IE	Firefox	Safari	JRE
DEC	1.5738	2.3046	1.6381	0.9926	0.4471	0.9401
χ_c^2	19.6751	19.6751	19.6751	19.6751	19.6751	19.6751
χ_s^2	194.1639	190.0132	1175.0693	1146.9044	1025.0824	784.2028
p value	1.20641E−35	8.7302E−35	3.8201E−245	4.4737E−239	7.6726E−213	4.6663E−161

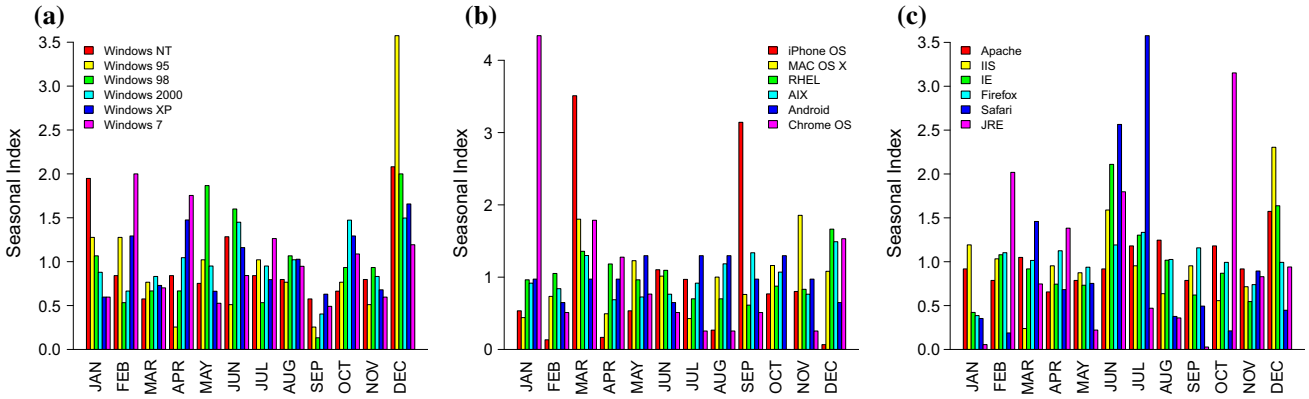

Fig. 3 Seasonal indices of Windows OSes, non-Windows OSes and Web-related software systems. Individual values are in Table 2. **a** Windows OSes. **b** non-Windows OSes. **c** Web-related software

Table 3 ANOVA table for seasonal indices from Windows OSes

Windows	SS	*df*	MS	*F*	*p* value	F_{crit}
Between groups	9.314879458	11	0.846807223	4.655838669	4.16254E−05	1.952211939
Within groups	10.91284235	60	0.181880706			
Total	20.22772181	71				

[13]. To evaluate the significance of the non-uniformity of the distribution among the seasonal indices, we applied the Chi-square test to the grand total of each month against the mean value (total vulnerabilities divided by 12). In this paper, the level of alpha chosen is 0.05. Hence, when the p value of the Chi-square test is below 0.05, the null hypothesis that there is no seasonality in the dataset will be rejected. In Table 2, we see that the systems yield extremely small p values, thereby providing strong evidence of the non-uniformity of the distributions of vulnerability discovery rates contrary to the null hypothesis.

To determine which months actually have statistically significant higher or lower indices, the ANOVA test along with the Fisher's LSD test are conducted on the mean index values from each month grouped by software categories. As observed previously, since the ANOVA test can only tell whether the mean index values among the 12 months are the same or not, Fisher's LSD test is also conducted after confirming the unequal performance via the ANOVA test.

Tables 3, 4 and 5 show ANOVA tables for each software group. Here, the alpha level is 0.05 for the F-test. To be statistically significant, the F value needs to be greater than the corresponding F critical with sufficiently small p values (less than 0.05). In Tables 3 and 5, F values greater than the F critical value, confirm that not all the months have equal mean values whereas ANOVA test for the non-Windows OSes does not produce statistically significant value. In addition, the F value from the Windows OSes is larger than the one from the Web-related software, implying that seasonal fluctuations in Windows OSes are more dynamic.

Tables 6, 7 and 8 give the absolute differences among the seasonal indices and the significance of pairwise comparisons, with italicized cells representing statistically significant differences. To be a italicized cell, differences between two compared mean values need to be greater than the corresponding calculated LSD value. The LSD values for each table are $\text{LSD}_{\text{Windows}} = 0.4924$, $\text{LSD}_{\text{non-Windows}} = 0.7901$ and $\text{LSD}_{\text{Web}} = 0.6811$, respectively.

For the seasonal index values, Table 6 confirms that, in Windows OSes, December is greater than all the other months and September is less than January, February, April, May, June, and August. Table 7 confirms that, in non-Windows

Table 4 ANOVA table for seasonal indices from non-Windows OSes

Non-windows	SS	*df*	MS	*F*	*p* value	F_{crit}
Between groups	6.761092828	11	0.614644803	1.312574578	0.240035623	1.952211939
Within groups	28.09645163	60	0.468274194			
Total	34.85754446	71				

Table 5 ANOVA table for seasonal indices from Web-related software

Web	SS	*df*	MS	*F*	*p* value	F_{crit}
Between groups	7.988696315	11	0.72624512	2.087102541	0.035136421	1.952211939
Within groups	20.8780864	60	0.347968107			
Total	28.86678272	71				

Table 6 LSD test for Windows seasonal index; $LSD_{Windows} = 0.4924$

Month	Mean	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
		1.0607	1.1016	0.7118	1.0064	0.9634	1.1411	0.9009	0.9377	0.4149	1.0362	0.7248	2.0005
JAN	1.0607	0	0.0409	0.3488	0.0543	0.0972	0.0804	0.1598	0.1229	*0.6458*	0.0244	0.3359	*0.9399*
FEB	1.1016		0	0.3897	0.0952	0.1382	0.0395	0.2007	0.1638	*0.6867*	0.0654	0.3768	*0.8989*
MAR	0.7118			0	0.2946	0.2516	0.4293	0.1890	0.2259	0.2969	0.3244	0.0130	*1.2887*
APR	1.0064				0	0.0430	0.1347	0.1055	0.0687	*0.5915*	0.0298	0.2816	*0.9941*
MAY	0.9634					0	0.1777	0.0626	0.0257	*0.5485*	0.0728	0.2386	*1.0371*
JUN	1.1411						0	0.2402	0.2034	*0.7262*	0.1049	0.4163	*0.8594*
JUL	0.9009							0	0.0369	0.4860	0.1354	0.1761	*1.0997*
AUG	0.9377								0	*0.5229*	0.0985	0.2129	*1.0628*
SEP	0.4149									0	0.6213	0.3099	*1.5856*
OCT	1.0362										0	0.3114	*0.9643*
NOV	0.7248											0	*1.2757*
DEC	2.0005												0

Table 7 LSD test for non-Windows seasonal index; $LSD_{non\text{-}Windows} = 0.7901$

Month	Mean	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
		1.3616	0.6532	1.7882	0.7968	0.9193	0.8560	0.7612	0.7844	1.2229	0.8622	0.9137	1.0805
JAN	1.3616	0	0.7084	0.4266	0.5647	0.4423	0.5056	0.6004	0.5771	0.1387	0.4993	0.4478	0.2810
FEB	0.6532		0	*1.1350*	0.1437	0.2661	0.2028	0.1080	0.1313	0.5697	0.2091	0.2606	0.4274
MAR	1.7882			0	*0.9913*	*0.8689*	*0.9322*	*1.0270*	*1.0037*	0.5653	*0.9259*	*0.8744*	0.7076
APR	0.7968				0	0.1225	0.0592	0.0357	0.0124	0.4260	0.0654	0.1169	0.2837
MAY	0.9193					0	0.0633	0.1581	0.1349	0.3036	0.0571	0.0056	0.1613
JUN	0.8560						0	0.0948	0.0716	0.3669	0.0062	0.0577	0.2245
JUL	0.7612							0	0.0233	0.4617	0.1011	0.1526	0.3194
AUG	0.7844								0	0.4384	0.0778	0.1293	0.2961
SEP	1.2229									0	0.3606	0.3091	0.1423
OCT	0.8622										0	0.0515	0.2183
NOV	0.9137											0	0.1668
DEC	1.0805												0

Table 8 LSD test for Web-related software seasonal index; $LSD_{Web} = 0.6811$

Month	Mean	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
		0.5544	1.0349	0.9043	0.9240	0.7175	1.6950	1.4697	0.7768	0.6735	1.1603	0.7736	1.3160
JAN	0.5544	0	0.4805	0.3499	0.3696	0.1631	*1.1406*	*0.9153*	0.2224	0.1191	0.6059	0.2193	*0.7617*
FEB	1.0349		0	0.1305	0.1109	0.3174	0.6602	0.4348	0.2581	0.3614	0.1254	0.2612	0.2812
MAR	0.9043			0	0.0196	0.1869	*0.7907*	0.5653	0.1275	0.2309	0.2560	0.1307	0.4117
APR	0.9240				0	0.2065	*0.7711*	0.5457	0.1471	0.2505	0.2363	0.1503	0.3921
MAY	0.7175					0	*0.9776*	*0.7522*	0.0593	0.0440	0.4428	0.0562	0.5986
JUN	1.6950						0	0.2254	*0.9182*	*1.0216*	0.5347	*0.9214*	0.3790
JUL	1.4697							0	*0.6929*	*0.7962*	0.3094	*0.6960*	0.1536
AUG	0.7768								0	0.1033	0.3835	0.0032	0.5392
SEP	0.6735									0	0.4868	0.1002	0.6426
OCT	1.1603										0	0.3867	0.1557
NOV	0.7736											0	0.5424
DEC	1.3160												0

OSes, March is greater than all the other months, except January, September, and December. Table 8 shows that June and July are greater than January, March, August, September, and November. Plus, June is also greater than April and May. Also, the table says December is greater than January.

5.2 Autocorrelation function analysis

The autocorrelation function (ACF) in time series analysis is calculated by computing the correlation between a variable value and the successive values of the same variable with some time lags. Thus, ACF measures the linear relationship between time series observations separated by a lag of k time units [5,30]. When an ACF value is located outside of defined confidence intervals at a lag k, there is a significant relationship associated with that time lag.

Tables 9, 10 and 11 show the ACF values with 95 % confidence intervals for the three software groups, respectively. In tables, the bold font indicates a value outside of confidence intervals, and superscripts represent time lags in month ranging from 0 to 23. For the Windows OSes, since the year-end period tends to have the majority of higher seasonal indices and September has particularly low values, we expect that lags corresponding to about 3 months or around its multiples would have their corresponding ACF values outside the confidence interval. In Table 9, we observe that for Windows NT, the lags for 0, 1, 2, 5, 6, 7, and 11 months are outside of confidence interval; in other words, there are strong autocorrelations with the lags that are multiples of around three or eleven, confirming a seasonal pattern.

For Windows 95, lags for 0, 3, 6, and 7 months; for Windows 2000, 0, 1, 2, 3, 4, 5 and 6 months; for Windows XP, 0, 2, 3, 4, 5, 6, 10, 14, 16, 18, and 22 months are significantly different from the zero of ACF which confirms a seasonal pattern. For Windows 98 and Windows 7, only the lag of 0 is outside the confidence interval. As we expected, roughly a 3 and 11 month periodicity is observed in most of the cases. The same approach was earlier applied in [30,34] to demonstrate seasonality in datasets pertaining to other research areas.

In Table 10, for iPhone OS, lags for 0, 6, and 18 are outside the confidence interval. For OSX, 0, 3, 4, 7, 8, 10, 11, 12, 15, 17, 18, 20, 21, 22, and 23 months; for RHEL, 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9; for AIX, 0, 1, 2, 3, 4, 5, 6, 7, 14, and 21; for Android, lags of 0 and 16 are outside the confidence interval, while Chrome OS has only one lag which is 0.

In Table 11, for the Apache Web server, lags of 0 and 7 are outside the confidence interval. For the IIS, lags of 0, 1, 4, 5, 6, 7, 8, 10, 11, 14, 18, 21, and 23 are outside the confidence interval. For Internet Explorer, lags of 0, 1, 2, 3, 4, and 8 are located outside the boundary. For Firefox, lags of 0, 2, 3, 11, and 14 are located outside the confidence interval. For the Safari web browser, only lags number 0 and 12 are outside. Interestingly, all the significant legs for JRE are multiples of four, 0, 4, 8, 12, 16, 20.

6 Seven-day periodicity in the institutional vulnerability scans

In this section, another periodic behavior related to software vulnerabilities with a much shorter periodicity is examined. We examine the available data for the presence of weekly periodic trends. Periodic scanning is a major part of corporate security strategy. Some security service vendors such as Qualys [27] collect large amount of data which is quite valuable because it comes from real systems in major industrial organizations. We have mined one such data collection to examine periodicity in the presence of unpatched vulnerabilities and exploitations in the case of a worm.

Table 9 Individual ACF values for Windows OSes

Windows NT; 95 % confidence interval = (−0.1414482, 0.1414482)											
1^{0}	**0.146**^{1}	**0.152**^{2}	0.07^{3}	0.084^{4}	**0.227**^{5}	**0.178**^{6}	**0.148**^{7}	0.017^{8}	0.001^{9}	0.041^{10}	**0.225**^{11}
0.015^{12}	0.115^{13}	0.08^{14}	0.036^{15}	−0.001^{16}	−0.019^{17}	0.045^{18}	0.016^{19}	0.011^{20}	−0.008^{21}	−0.03^{22}	−0.042^{23}
Windows 95; 95 % confidence interval = (−0.1569227, 0.1569227)											
1^{0}	0.115^{1}	0.146^{2}	**0.188**^{3}	0.112^{4}	0.143^{5}	**0.228**^{6}	**0.198**^{7}	0.068^{8}	0.142^{9}	0.152^{10}	0.076^{11}
0.066^{12}	**0.173**^{13}	0.092^{14}	0.073^{15}	0.04^{16}	0.093^{17}	−0.059^{18}	0.048^{19}	0.036^{20}	0.046^{21}	0.037^{22}	0.015^{23}
Windows 98; 95 % confidence interval = (−0.170593, 0.170593)											
1^{0}	0.166^{1}	0.154^{2}	0.061^{3}	−0.006^{4}	0.169^{5}	0.137^{6}	0.13^{7}	0.07^{8}	0.113^{9}	0.056^{10}	0.063^{11}
0.105^{12}	−0.004^{13}	−0.025^{14}	−0.039^{15}	−0.144^{16}	0.02^{17}	−0.04^{18}	−0.038^{19}	0.061^{20}	−0.103^{21}	0.029^{22}	−0.054^{23}
Windows 2000; 95 % confidence interval = (−0.1414482, 0.1414482)											
1^{0}	**0.163**^{1}	**0.294**^{2}	**0.162**^{3}	**0.165**^{4}	**0.311**^{5}	**0.206**^{6}	0.104^{7}	0.116^{8}	0.095^{9}	0.105^{10}	0.093^{11}
0.091^{12}	−0.003^{13}	0.047^{14}	−0.072^{15}	0.023^{16}	−0.006^{17}	0.096^{18}	−0.138^{19}	0.002^{20}	−0.118^{21}	−0.094^{22}	−0.017^{23}
Windows XP; 95 % confidence interval = (−0.1460871, 0.1460871)											
1^{0}	0.065^{1}	**0.256**^{2}	**0.16**^{3}	**0.173**^{4}	**0.165**^{5}	**0.281**^{6}	0.116^{7}	**0.217**^{8}	0.048^{9}	**0.149**^{10}	0.062^{11}
0.14^{12}	0.081^{13}	**0.178**^{14}	−0.053^{15}	**0.158**^{16}	−0.068^{17}	**0.179**^{18}	0.032^{19}	0.081^{20}	−0.02^{21}	**0.202**^{22}	−0.021^{23}
Windows 7; 95 % confidence interval = (−0.230984, 0.230984)											
1^{0}	0.012^{1}	0.118^{2}	0.14^{3}	0.154^{4}	−0.042^{5}	0.095^{6}	0.043^{7}	0.209^{8}	−0.095^{9}	0.011^{10}	−0.058^{11}
−0.062^{12}	−0.112^{13}	0.048^{14}	−0.069^{15}	−0.012^{16}	−0.139^{17}	−0.037^{18}	−0.002^{19}	−0.096^{20}	−0.11^{21}	0.221^{22}	−0.085^{23}

Table 10 Individual ACF values for non-Windows OSes

iPhone OS; 95 % confidence interval = (−0.2138496, 0.2138496)										
$\mathbf{1}^{0}$	-0.078^{1}	-0.068^{2}	-0.043^{3}	0.115^{4}	0.062^{5}	$\mathbf{0.28}^{6}$	-0.057^{7}	0.056^{8}	-0.047^{9}	0.203^{10}	-0.032^{11}
0.19^{12}	-0.058^{13}	0.068^{14}	-0.019^{15}	-0.018^{16}	-0.089^{17}	$\mathbf{0.46}^{18}$	-0.029^{19}	-0.081^{20}	-0.02^{21}	0.041^{22}	0.006^{23}
OSX; 95 % confidence interval = (−0.1372249, 0.1372249)										
$\mathbf{1}^{0}$	-0.006^{1}	-0.017^{2}	$\mathbf{0.274}^{3}$	$\mathbf{0.287}^{4}$	0.107^{5}	0.099^{6}	$\mathbf{0.284}^{7}$	$\mathbf{0.156}^{8}$	0.101^{9}	$\mathbf{0.15}^{10}$	$\mathbf{0.21}^{11}$
$\mathbf{0.203}^{12}$	0.094^{13}	0.13^{14}	$\mathbf{0.235}^{15}$	0.059^{16}	$\mathbf{0.166}^{17}$	$\mathbf{0.236}^{18}$	0.135^{19}	$\mathbf{0.167}^{20}$	$\mathbf{0.139}^{21}$	0.055^{22}	$\mathbf{0.153}^{23}$
RHEL; 95 % confidence interval = (−0.1333587, 0.1333587)										
$\mathbf{1}^{0}$	$\mathbf{0.617}^{1}$	$\mathbf{0.41}^{2}$	$\mathbf{0.431}^{3}$	$\mathbf{0.477}^{4}$	$\mathbf{0.289}^{5}$	$\mathbf{0.143}^{6}$	$\mathbf{0.213}^{7}$	$\mathbf{0.287}^{8}$	$\mathbf{0.143}^{9}$	0.016^{10}	0.068^{11}
0.063^{12}	0.017^{13}	-0.009^{14}	0.023^{15}	0.048^{16}	-0.004^{17}	-0.043^{18}	-0.044^{19}	-0.042^{20}	-0.013^{21}	-0.025^{22}	-0.029^{23}
AIX; 95 % confidence interval = (−0.1206274, 0.1206274)										
$\mathbf{1}^{0}$	$\mathbf{0.134}^{1}$	$\mathbf{0.194}^{2}$	$\mathbf{0.172}^{3}$	$\mathbf{0.172}^{4}$	$\mathbf{0.14}^{5}$	$\mathbf{0.186}^{6}$	$\mathbf{0.246}^{7}$	0.068^{8}	0.116^{9}	$\mathbf{0.131}^{10}$	0.066^{11}
$\mathbf{0.135}^{12}$	0.047^{13}	$\mathbf{0.151}^{14}$	0.021^{15}	0.103^{16}	0.011^{17}	0.078^{18}	0.099^{19}	0.001^{20}	$\mathbf{0.182}^{21}$	-0.057^{22}	0.037^{23}
Android; 95 % confidence interval = (−0.2530303, 0.2530303)										
$\mathbf{1}^{0}$	0.062^{1}	0.079^{2}	0.186^{3}	-0.093^{4}	-0.132^{5}	-0.104^{6}	0.038^{7}	-0.172^{8}	-0.109^{9}	0.148^{10}	-0.108^{11}
-0.182^{12}	0.119^{13}	-0.092^{14}	-0.029^{15}	$\mathbf{0.261}^{16}$	0.118^{17}	0.101^{18}	-0.018^{19}	0.101^{20}	-0.075^{21}	-0.059^{22}	0.004^{23}
Chrome OS; 95 % confidence interval = (−0.2828964, 0.2828964)										
$\mathbf{1}^{0}$	0.219^{1}	-0.013^{2}	-0.084^{3}	0.056^{4}	-0.066^{5}	-0.102^{6}	-0.101^{7}	0.007^{8}	-0.042^{9}	-0.041^{10}	0.036^{11}
-0.026^{12}	0.002^{13}	-0.032^{14}	-0.044^{15}	-0.049^{16}	0.003^{17}	0.004^{18}	0.043^{19}	-0.044^{20}	-0.038^{21}	-0.05^{22}	-0.012^{23}

Table 11 Individual ACF values for Web-related software

Apache web server; 95 % confidence interval = (−0.1333587, 0.1333587)											
1^{0}	0.033^{1}	0.122^{2}	0.055^{3}	0.131^{4}	0.08^{5}	0.014^{6}	**0.189**^{7}	0.045^{8}	0.08^{9}	0.022^{10}	0.061^{11}
0.052^{12}	−0.04^{13}	0.036^{14}	0.04^{15}	−0.034^{16}	0.04^{17}	0.118^{18}	−0.059^{19}	0.122^{20}	0.033^{21}	0.121^{22}	−0.016^{23}
IIS; 95 % confidence interval = (−0.1333587, 0.1333587)											
1^{0}	**0.17**^{1}	0.049^{2}	0.113^{3}	**0.208**^{4}	**0.152**^{5}	**0.197**^{6}	**0.384**^{7}	**0.201**^{8}	0.049^{9}	**0.202**^{10}	**0.218**^{11}
0.114^{12}	0.072^{13}	**0.27**^{14}	0.06^{15}	0.116^{16}	0.123^{17}	**0.313**^{18}	0.006^{19}	−0.032^{20}	**0.142**^{21}	0.06^{22}	**0.167**^{23}
IE; 95 % confidence interval = (−0.1372249 , 0.1372249)											
1^{0}	**0.253**^{1}	**0.151**^{2}	**0.162**^{3}	**0.297**^{4}	0.102^{5}	0.107^{6}	0.135^{7}	**0.164**^{8}	0.106^{9}	0.108^{10}	0.132^{11}
0.283^{12}	0.074^{13}	0.039^{14}	0.044^{15}	0.063^{16}	−0.014^{17}	0.036^{18}	0.006^{19}	0.028^{20}	−0.037^{21}	0.013^{22}	−0.031^{23}
Firefox; 95 % confidence interval = (−0.170593, 0.170593)											
1^{0}	0.058^{1}	**0.224**^{2}	**0.339**^{3}	0.137^{4}	0.133^{5}	0.091^{6}	0.159^{7}	0.099^{8}	0.148^{9}	0.148^{10}	**0.242**^{11}
0.039^{12}	0.085^{13}	**0.235**^{14}	−0.013^{15}	−0.003^{16}	0.161^{17}	−0.026^{18}	0^{19}	0.081^{20}	−0.002^{21}	−0.01^{22}	−0.053^{23}
Safari; 95 % confidence interval = (−0.170593, 0.170593)											
1^{0}	0.01^{1}	−0.019^{2}	0.063^{3}	0.073^{4}	0.012^{5}	−0.081^{6}	−0.039^{7}	0.049^{8}	0.081^{9}	−0.061^{10}	−0.014^{11}
0.335^{12}	0.086^{13}	−0.042^{14}	−0.039^{15}	0.089^{16}	0.05^{17}	−0.082^{18}	−0.049^{19}	0.082^{20}	0.072^{21}	0.067^{22}	−0.042^{23}
JRE; 95 % confidence interval = (−0.1633303, 0.1633303)											
1^{0}	−0.126^{1}	0.063^{2}	−0.066^{3}	**0.504**^{4}	0.025^{5}	0.142^{6}	−0.112^{7}	**0.455**^{8}	−0.06^{9}	0.018^{10}	0.057^{11}
0.264^{12}	−0.055^{13}	0.051^{14}	−0.03^{15}	**0.306**^{16}	−0.053^{17}	0.042^{18}	−0.004^{19}	**0.242**^{20}	−0.046^{21}	0.093^{22}	−0.018^{23}

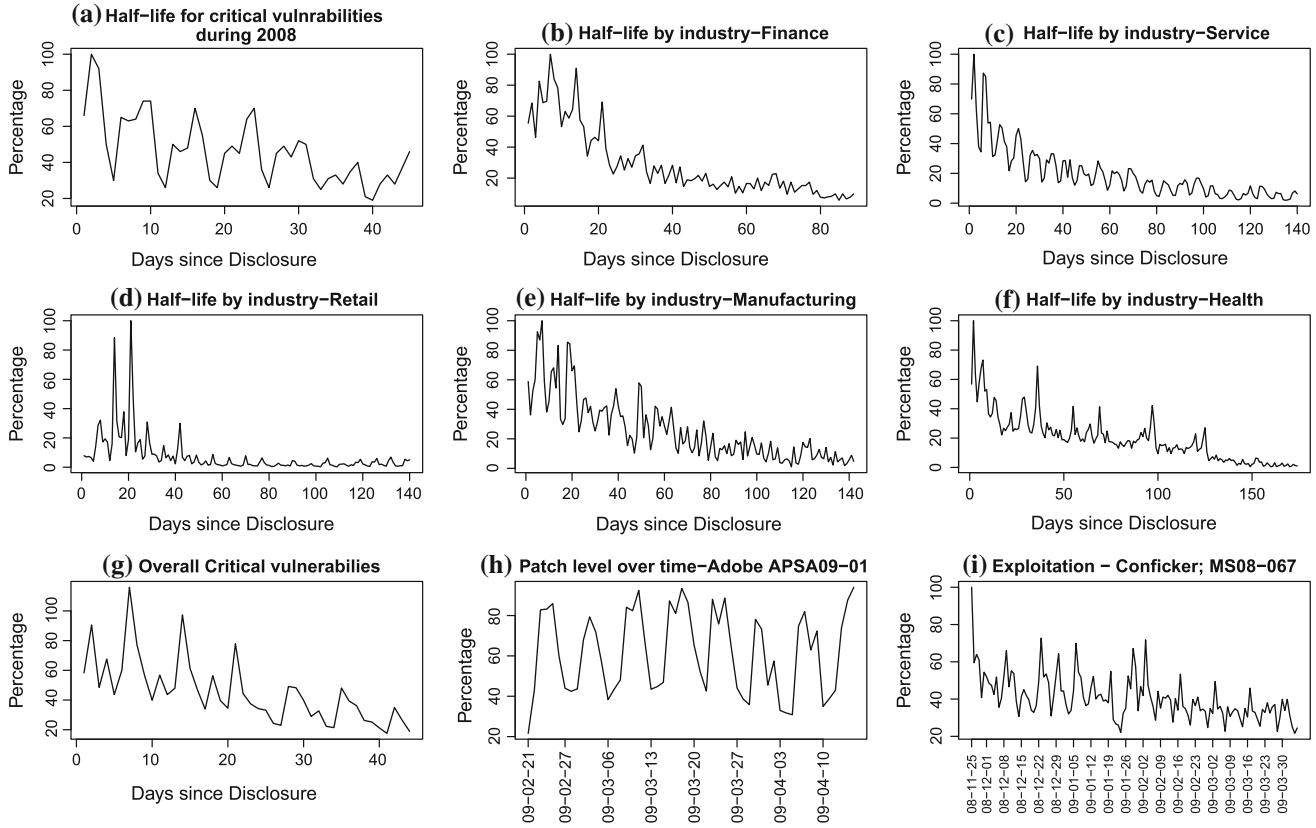

Fig. 4 Run charts for the seven half-life plots (critical vulnerabilities during 2008, by industries of finance, service, retail, manufacturing, and health, and overall critical vulnerabilities), patch level and exploitation

Qualys has been involved in collecting and plotting such data for several years. In a 2009 report [27], the organization presented the data collected during 2008, which represents 104 million global vulnerability scans, including 82 million internal scans and 22 million external Internet-based scans. The data demonstrates the encountering of more than 72 million critical vulnerabilities among the 680 million detections. About 3500 organizations that represented major industry sectors of Financial, Health, Manufacturing, Service, and Wholesale/Retail were scanned worldwide. Four distinct and quantifiable attributes related to software vulnerabilities were introduced by the company:

- Half-life: the time interval measuring the reduction by half of a vulnerability's occurrence. Since a shorter half-life indicates faster remediation, over time, this metric demonstrates how successful efforts have been to eradicate vulnerabilities.
- Prevalence: the turnover rate of vulnerabilities in the "Top 20" list during a year. The prevalence of such vulnerabilities are dangerous because they represent ongoing potent risks to computing environments. Risk rises as the prevalence rate increases because of the larger total number of the top 20 risks tallied during a year.
- Persistence: the measure of the total life span of vulnerabilities. The fact that vulnerabilities persist and do not conclusively die off is a red flag for security administrators. It underscores the importance of patching all systems and ensuring that old vulnerabilities are not inadvertently installed on new systems.
- Exploitation: the time interval between an exploit announcement and the first attack. This metric indicates the probable reaction time prior to the discovery of the method of exploiting the vulnerability. The worst scenario is a "zero day" attack because there is no reaction time.

Qualys terms the above four attributes "the Laws of Vulnerabilities". At the 2009 Black Hat USA conference (http://www.blackhat.com/html/bh-usa-09/bh-us-09-main.html), the following observations for the each law were presented. The average duration of a vulnerability's half-life is about 30 days, varying by industry sector. Prevalence has increased, with 60 remaining in the list in 2008 compared to 50 in 2004. Persistence remains virtually unlimited. Exploitation now occurs more rapidly, often within <10 days compared to 60 days in 2004.

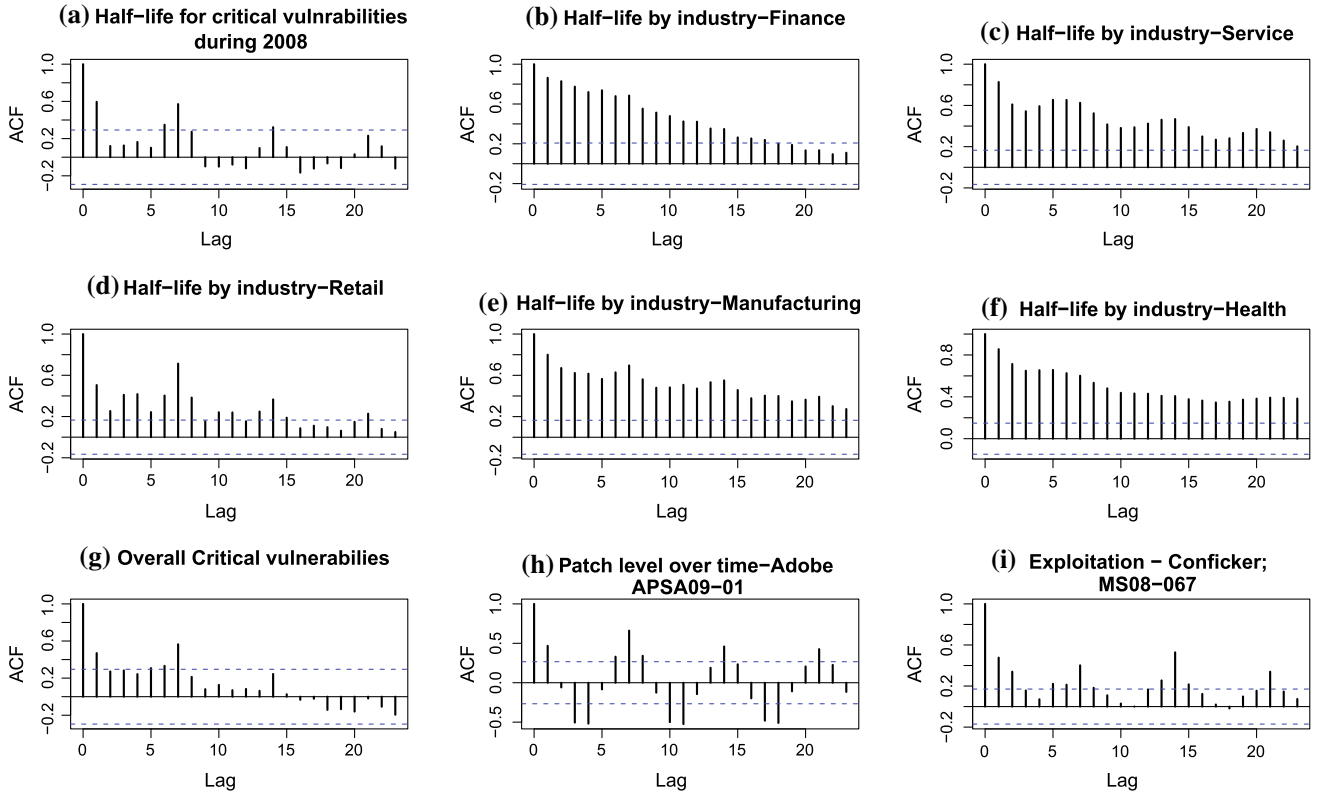

Fig. 5 Autocorrelation functions corresponding to the plots from Fig. 4. The *dashed lines* represent 95 % of confidence intervals. *Legs* are in day

Table 12 Weekly seasonal index values

Label (Fig. 4)	Day1 (Sun)	Day2 (Mon)	Day3 (Tue)	Day4 (Wed)	Day5 (Thu)	Day6 (Fri)	Day7 (Sat)	$\chi^2_{\text{statistic}}$	p value
(a)	1.0494	1.4099	1.3600	0.7210	0.5425	0.9423	0.9745	165.6114	3.83E−33
(b)	0.9825	1.0672	0.7819	1.0322	0.9973	0.8794	1.2592)	49.029	7.36E−09
(c)	1.2194	1.0796	0.6290	0.6312	0.9974	1.2643	1.1788	165.0925	4.94E−33
(d)	1.1784	0.6445	0.6811	0.8007	0.3960	0.7976	2.5014	435.1142	7.84E−91
(e)	1.0571	0.7047	0.8268	1.2247	0.9567	0.9398	1.2899	135.1223	1.07E−26
(f)	1.1573	1.0117	0.8913	0.8334	0.9209	1.1374	1.0477	44.6814	5.41E−08
(g)	1.1848	1.1069	0.7631	0.9798	0.7022	0.7370	1.5258	148.7978	1.39E−29
(h)	0.6758	1.3090	1.2945	1.2569	1.0805	0.7046	0.6783	236.8411	2.65E−48
(i)	0.9559	1.0068	1.2973	1.0203	1.0353	0.9534	0.7307	119.9789	1.65E−23

We observed that in the report, most of the plots visually suggest a short-term 7-day periodicity, as shown in Fig. 4. This section examines the statistical significance of the periodic pattern of selected plots in the report by using the seasonal index and autocorrelation analysis.

Figure 4 shows nine run charts from the report plotted daily. The plots are normalized using the maximum value set as 100 . Even visually, it is clearly observed that there are certain periodic patterns in the data. The values decline as the result of a remediation and increase due to new installations. Figure 5 displays corresponding ACF values from Fig. 4. In the figure, lags of seven (or its multiples) tend to either have higher values or values outside the ±95 % confidence intervals shown by the dashed lines. This demonstrates strong autocorrelations with lags that are multiples of seven, which confirms a 7-day periodicity in the data.

Table 12 shows the calculated weekly seasonal index values from Fig. 4 with Chi-square test. Since there is no information as to day of the week except (h) and (i) from Fig. 4, it is labeled as day1, day2, . . ., day7, while the two cases are mentioned on specific weekdays (Sun. through Sat.). From (a) to (g), it is observed that values generally tend to be clustered into high or low values consecutively. For example, in (g), higher values appear on day7, day1,

and day2 successively. For (h) and (i), weekdays (Monday–Thursday) tend to have higher index values for the number of incidents. The observed patterns could be related to software vendors' patch release policies, organizations' patch management strategies, or the behavior of specific individual. To be statistically significant for the calculated seasonal index values, the Chi-square statistic values must be greater than the corresponding critical values with a sufficiently small p value. In the table, the small p values confirm the non-uniform distributions.

7 Possible factors causing periodic behaviors

Seasonality in many natural biological systems can be easily explained in terms of the annual seasons due to the rotation of the earth. Sometimes the causes of seasonality may be harder to pin down. For vulnerabilities, Rescorla [29] has mentioned a possible cause for the observed year-end seasonality, suggesting that the large number of vulnerabilities reported at the year-end may be a result of the end-of-year cleanup. He did not, however, discuss this possibility in detail. One possibility is that it may be related to year-end reports which many organizations require to be completed before the year's end.

Further research is needed to determine why the discovery of vulnerabilities in Microsoft products tends to peak in the mid-year months in addition to the year-end months. One possibility is that Defcon (https://www.defcon.org/main.html), a major computer security-related conference that originated in 1993 takes place mainly in July or August. While it originally stated as a hackers convention, it is attended by a large number of security professionals. The potential conference participants might have a higher incentive [4] to find the vulnerabilities before the conference, to brag about, especially in popular Microsoft products.

Figure 6a shows the number of occurrences of Defcon and Black Hat; the two best known conventions at which security vulnerabilities are announced. At the same time, the August–November period appears to be associated with the release of a large number of new Microsoft products. Figure 6b shows the products' release months for major versions of Windows OSes and Internet Explorer. The major versions of Windows and Internet Explorer tend to be released during the period between June and November. This may be related to the starting of school or to Christmas and New Year shopping seasons, when many people buy new computers with new operating systems, known as IT seasonality. Condon et al. [9] also observed that occurrences of software security incidents increase during the academic calendar, and the most relevant form of institutional type of seasonality is summer vacation from school [19]. In December, emphasis may shift to identifying and handling vulnerabilities.

The reason for similar seasonality for the Windows operating systems, IIS, and Internet Explorer might be due to the fact that IIS and Internet Explorer are distributed only for the Windows platform, while vulnerabilities of Web servers and browsers may be correlated to parent operating system platforms.

Figure 7a, b shows the number of vulnerabilities grouped by day of the week in terms of disclosure date from OSVDB (data on 2010-10-06; relatively old (·.·) stop providing dump data) and published date from NVD (data on 2014-08-05), respectively. "Disclosure date" refers to the date on which vulnerabilities are publicly disclosed, whereas "published

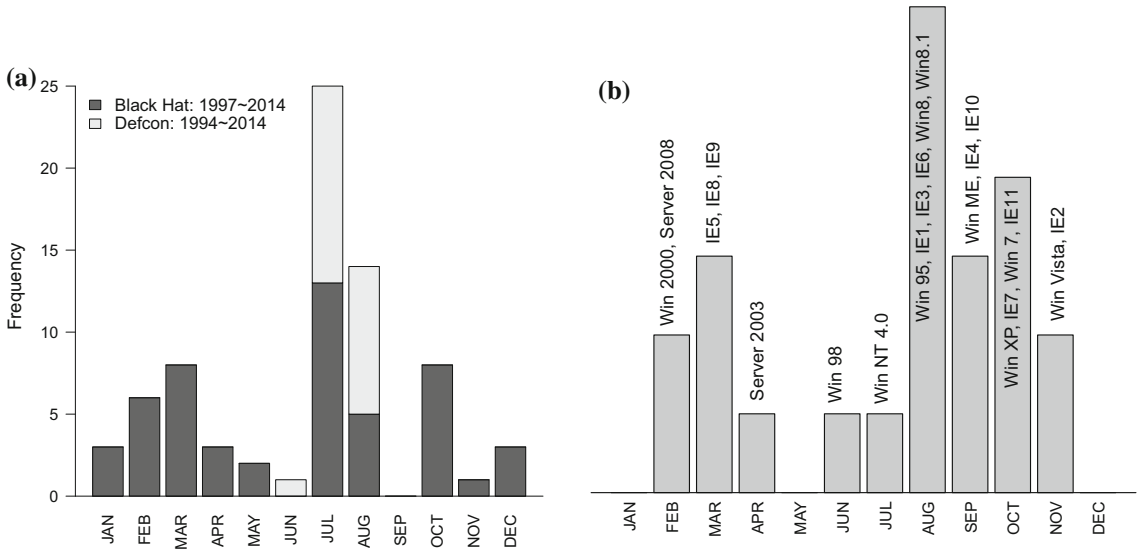

Fig. 6 Frequency of Black Hat and Defcon by month, and major Microsoft software system release time by month. **a** Black Hat and Defcon by month. **b** MS release by month

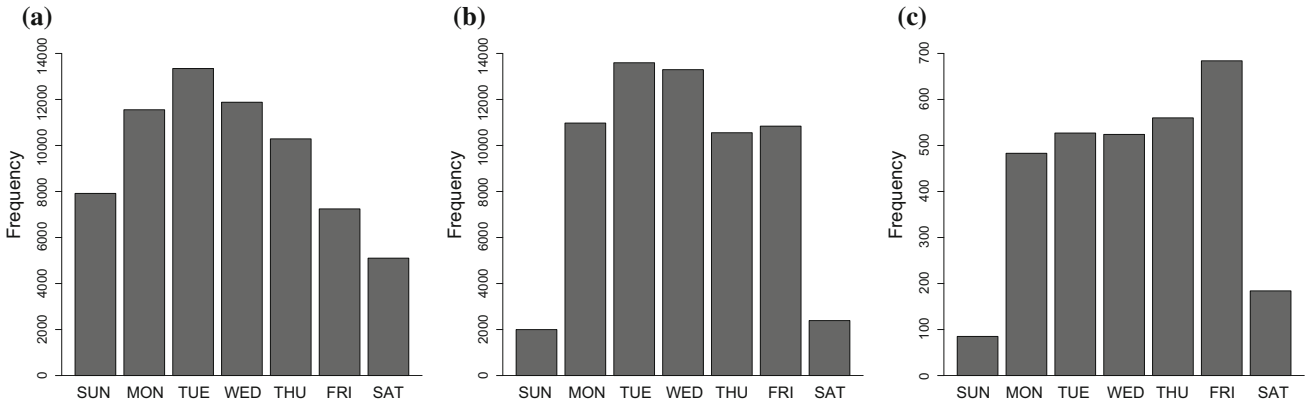

Fig. 7 Frequency of vulnerability disclosure date, published date, and data loss report data by day of the week. **a** Disclosure date (https://blog.osvdb.org) no. of vuln.: 67325. **b** Published date (https://nvd.nist.gov) no. of vul.: 63647. **c** Report date (https://blog.datalossdb.org) no. of reports: 3047

date" is the date on which vulnerabilities are published on the database according to OSVDB and NVD, respectively; technically, the two dates have the same meaning. It is observed that both the disclosure date's and the published date's values peak on Tuesday, values generally increasing as Tuesday approaches and decreasing thereafter. Figure 7c represents the number of data loss incidents reported by organizations in terms of day of the week by the Open Security Foundation (data on 2010-12-28; relatively old (·.·) stop providing dump data). Although the data does not directly reflect vulnerability information, it clearly epitomizes the weekday versus weekend phenomenon.

Anbalagan and Vouk [2] suggest a possible weekly pattern for fixing ordinary defects. Those reported on Tuesdays tend to be fixed faster. Their graph displaying the average correction time shows an opposite pattern to the first two plots in Fig. 7, i.e., values decreasing toward Tuesday and increasing after that point. This could be because developers know from experience that they require a higher level of effort on that day; consequently, in order not to fall behind, more effort is done on Tuesdays.

The findings of annual seasonality in this paper are supported by the results of a survey conducted by Tufin Technologies (http://www.net-security.org/secworld.php?id=7928) related to hackers' habits, with information provided by seventy-nine hackers attending the Defcon 17 conference in 2009. Analysis of the survey reveals that Christmas and New Year holiday seasons are popular with hackers targeting Western countries and that hackers spend time hacking on weekdays rather than weekends. Here are some numbers from the survey related to our study:

- 89 said taking a summer vacation would have little impact on their hacking activities.
- 81 said they are far more active during the winter holidays, with 56 citing Christmas as the best time for hacking and 25 naming New Year's Eve.
- 52 said that they hack other systems during weekday evenings; 32 said that their operations take place during weekday working hours; and only 15 of hackers do their hacking on weekends.

8 Conclusions

Analysis of the vulnerability data using seasonal index and autocorrelation function approaches shows that there is indeed a statistically significant annual and weekly periodic pattern in software vulnerability-related activities. In the first part of this paper, for all three software groups examined, a higher vulnerability discovery rate is encountered in certain months. In Microsoft products, a higher incidence during the mid-year periods is also observed. Also, 7-day periodic behavior was observed in the vulnerability scan data; higher activity during weekdays than weekends has been confirmed. Specifically, vulnerability activity values corresponding to Tuesday tend to be higher than other days of the week.

One of the main contributions of this paper is that it provides a variety of evidence that there actually do exist short and long term seasonal patterns which have been but vaguely recognized among security researchers thus far. The results found in this paper should be used to optimize resource allocations, patch management practices, and on the general determination of IT-related risks. For example, system administrators should apply patches prior to the time when seasonal indices are relatively high for both short and long term strategies.

Further work is needed to develop methods for the prediction of future vulnerability discovery trends using the Box–Jenkins time series Model (ARIMA), which uses autocorrelation function, periodogram, spectral analysis, and partial autocorrelation function analysis. Chen et al. [8] have already taken a periodic factor into their multi-cycle vulnerability discovery model. However, their periodic factor

does not directly consider long or short term seasonality, but rather the vulnerability discovery rate. The findings in this paper may be used in conjunction with the longer-term trends described by the vulnerability discovery models to improve vulnerability discovery predictions and to optimize resource allocation.

References

1. Alhazmi, O.H., Malaiya, Y.K.: Application of vulnerability discovery models to major operating systems. IEEE Trans. Reliab. **57**(1), 14–22 (2008)
2. Anbalagan, P., Vouk, M.: "Days of the week" effect in predicting the time taken to fix defects. In: DEFECTS'09: Proceedings of the 2nd International Workshop on Defects in Large Software Systems, pp. 29–30, New York, NY, USA. ACM (2009)
3. Anderson, R: Security in open versus closed systems—the dance of boltzmann, coase and moore. In: Conference on Open Source Software, Economics, Law and Policy, pp. 1–15 (2002)
4. Arora, A., Telang, R.: Economics of software vulnerability disclosure. IEEE Secur. Priv. **3**(1), 20–25 (2005)
5. Bowerman, B.L., O'connell, R.T.: Time Series Forecsting: Unified Concepts and Computer Implementation, 2nd edn. Duxbury Press, Boston (1987)
6. Bozorgi, M., Saul, L.K., Savage, S., Voelker, G.M.: Beyond heuristics: learning to classify vulnerabilities and predict exploits. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'10, pp. 105–114, New York, NY, USA. ACM (2010)
7. Carrion-Baralt, J.R., Smith, C.J., Rossy-Fullana, E., Lewis-Femandez, R., Davis, K.L., Silverman, J.M.: Seasonality effects on schizophrenic births in multiplex families in a tropical island. Psychiatry Res. **142**(1), 93–97 (2006)
8. Chen, K., Feng, D.-G., Su, P.-R., Nie, C.-J., Zhang, X.-F.: Multi-cycle vulnerability discovery model for prediction. J. Softw. **21**(9), 2367–2375 (2010)
9. Condon, E., He, A., Cukier, M.: Analysis of computer security incident data using time series models. In: ISSRE'08: Proceedings of the 2008 19th International Symposium on Software Reliability Engineering, pp. 77–86, Washington, DC, USA. IEEE Computer Society (2008)
10. Eick, S.G., Graves, T.L., Karr, A.F., Marron, J.S., Mockus, A.: Does code decay? Assessing the evidence from change management data. IEEE Trans. Softw. Eng. **27**(1), 1–12 (2001)
11. Goonatilake, R., Herath, A., Herath, S., Herath, S., Herath, J.: Intrusion detection using the chi-square goodness-of-fit test for information assurance, network, forensics and software security. J. Comput. Small Coll. **23**, 255–263 (2007)
12. Heston, S.L., Sadka, R.: Seasonality in the cross-section of stock returns. J. Financ. Econ. **87**(2), 418–445 (2008)
13. Jaquith, A.: Security Metrics: Replacing Fear, Uncertainty and Doubt. Addison-Wesley Professional, Boston (2007)
14. Jegadeesh, N.: Evidence of predictable behavior of security returns. J. Finance **45**(3), 881-98 (1990)
15. Joh, H., Chaichana, S., Malaiya, Y.K.: Short-term periodicity in security vulnerability activity. In: International Symposium on Software Reliability Engineering, pp. 408–409 (2010)
16. Joh, H., Malaiya, Y. K.: Seasonal variation in the vulnerability discovery process. In: ICST'09: International Conference on Software Testing, Verification, and Validation, pp. 191–200, Los Alamitos, CA, USA. IEEE Computer Society (2009)
17. Joh, H., Malaiya, Y.K.: Modeling skewness in vulnerability discovery. Qual. Reliab. Eng. Int. **30**(8), 1445–1459 (2014). doi:10.1002/qre.1567
18. Kim, J., Malaiya, Y.K., Ray, I.: Vulnerability discovery in multi-version software systems. In: HASE'07: Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium, pp. 141–148, Washington, DC, USA. IEEE Computer Society (2007)
19. Koc, E., Altinay, G.: An analysis of seasonality in monthly per person tourist spending in Turkish inbound tourism from a market segmentation perspective. Tour. Manag. **28**(1), 227–237 (2007)
20. Kozina, M., Golub, M., Groš, S.: A method for identifying web applications. Int. J. Inf. Secur. **8**(6), 455–467 (2009)
21. Maes, J., Van Damme, S., Meire, P., Ollevier, F.: Statistical modeling of seasonal and environmental influences on the population dynamics of an estuarine fish community. Mar. Biol. **145**, 1033–1042 (2004)
22. Massacci, F., Nguyen, V.H.: Which is the Right Source for Vulnerability Studies? An Empirical Analysis on Mozilla Firefox. Technical report. University of Trento, Italy (2010)
23. Ott, R.L., Longnecker, M.T.: An Introduction to Statistical Methods and Data Analysis, 5th edn. Duxbury press, North Scituate (2000)
24. Ozment, A.: Improving vulnerability discovery models. In: QoP'07: Proceedings of the 2007 ACM Workshop on Quality of Protection, pp. 6–11, New York, NY, USA. ACM (2007)
25. Ozment, A., Schechter, S.E.: Milk or wine: does software security improve with age? In: USENIX-SS'06: Proceedings of the 15th Conference on USENIX Security Symposium, Berkeley, CA, USA. USENIX Association (2006)
26. Pfleeger, C.P., Pfleeger, S.L.: Security in Computing, 3rd edn. Prentice Hall PTR, Upper Saddle River (2003)
27. Qualys, I.: The laws of vulnerabilities 2.0. In Black Hat 2009, Presented by Wolfgang Kandek (CTO) (July 28, 2009)
28. Rescorla, E.: Security holes. who cares? In: SSYM'03: Proceedings of the 12th Conference on USENIX Security Symposium, pp. 75–90, Berkeley, CA, USA. USENIX Association (2003)
29. Rescorla, E.: Is finding security holes a good idea? IEEE Secur. Priv. **3**, 14–19 (2005)
30. Rios, M., Garcia, J.M., Sanchez, J.A., Perez, D.: A statistical analysis of the seasonality in pulmonary tuberculosis. Eur. J. Epidemiol. **16**(5), 483-8 (2000)
31. Romanov, A., Tsubaki, H., Okamoto, E.: An approach to perform quantitative information security risk assessment in it landscapes. JIP **18**, 213–226 (2010)
32. Salehian, A.: Arima time series modeling for forecasting thermal rating of transmission lines. In: Transmission and Distribution Conference and Exposition, 2003 IEEE PES, vol. 3, pp. 875–879 (2003)
33. Symantec. Symantec global internet security threat report: trends for 2009, vol. XV (2010)
34. Tran, N., Reed, D.: Automatic arima time series modeling for adaptive i/o prefetching. IEEE Trans. Parallel Distrib. Syst. **15**(4), 362–377 (2004)
35. Zhang, Z., Zheng, X., Zeng, D., Cui, K., Luo, C., He, S., Leischow, S.: Discovering seasonal patterns of smoking behavior using online search information. In: Intelligence and Security Informatics (ISI), 2013 IEEE International Conference on, pp. 371–373 (2013)