available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Measuring, analyzing and predicting security vulnerabilities in software systems

*O.H. Alhazmi, Y.K. Malaiya, I. Ray**

Department of Computer Science, Colorado State University, Fort Collins, CO 80523, United States

ARTICLE INFO

Article history:
Received 7 November 2005
Accepted 4 October 2006

Keywords:
Vulnerabilities
Security holes
Risk evaluation
Quantitative security modeling
Defect density

ABSTRACT

In this work we examine the feasibility of quantitatively characterizing some aspects of security. In particular, we investigate if it is possible to predict the number of vulnerabilities that can potentially be present in a software system but may not have been found yet. We use several major operating systems as representatives of complex software systems. The data on vulnerabilities discovered in these systems are analyzed. We examine the results to determine if the density of vulnerabilities in a program is a useful measure. We also address the question about what fraction of software defects are security related, i.e., are vulnerabilities. We examine the dynamics of vulnerability discovery hypothesizing that it may lead us to an estimate of the magnitude of the undiscovered vulnerabilities still present in the system. We consider the vulnerability discovery rate to see if models can be developed to project future trends. Finally, we use the data for both commercial and open-source systems to determine whether the key observations are generally applicable. Our results indicate that the values of vulnerability densities fall within a range of values, just like the commonly used measure of defect density for general defects. Our examination also reveals that it is possible to model the vulnerability discovery using a logistic model that can sometimes be approximated by a linear model.

1. Introduction

The security of software systems has been under considerable scrutiny in recent years. However, much of the work on security has been qualitative, focused on detection and prevention of vulnerabilities in these systems. There is a need to develop a perspective on the problem so that methods can be developed to allow security related risks to be evaluated quantitatively.

Quantitative methods can permit resource allocation for achieving a desired security level, as is done for software or system reliability. They can be used to evaluate metrics that can guide the allocation of resources for security testing, development of security patches and scheduling their releases. They can also be used by end-users to assess risks and estimate the needed redundancy in resources and procedures for handling potential security breaches. Unfortunately, only limited attention has been paid so far to the quantitative aspects of security. To develop quantitative methods for characterizing and managing security, we need to identify metrics that can be evaluated in practice and have a clearly defined interpretation. In this work, we examine the data on vulnerabilities identified in several popular operating systems to determine if concepts such as ''vulnerability density'' and ''vulnerability discovery rate'' can be characterized as useful metrics for software security. *Vulnerability* has been defined

* *Corresponding author*. Tel.: +1 970 4917097; fax: +1 970 4912466.
E-mail addresses: omar@cs.colostate.edu (O.H. Alhazmi), malaiya@cs.colostate.edu (Y.K. Malaiya), indrajit@cs.colostate.edu (I. Ray).
0167-4048/$ – see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2006.10.002

as ''a defect, which enables an attacker to bypass security measures'' (Schultz et al., 1990). Pfleeger (1997) defines vulnerability as ''a weakness in the security system that might be exploited to cause loss or harm''. Quantitative methods for general defects are now widely used to evaluate and manage overall software reliability. Since software system vulnerabilities—the faults associated with maintaining security requirements—can be considered a special case of software defects, similar measures for estimating security vulnerabilities appear useful.

It is not possible to guarantee absence of defects in nontrivial sized programs like operating systems. While extensive testing can isolate a large fraction of the defects, it is impossible to eliminate them. This is because the effort needed to discover residual defects increases exponentially (Lyu, 1995). Nonetheless, examination of defect densities (that is the number of defects identified in the unit size of the software code) is useful. It can lead to the identification of fault-prone modules that need special attention. Researchers have evaluated the ranges of defect densities typically encountered during different phases of the software life cycle using data from available sources (Musa et al., 1987). This has led to industry wide standards for software defect densities. The information can be used for comparison with the defect density measured in a project at a specific phase. The result can identify if there is a need for further testing or process improvement. Similar methods for managing the security aspects of systems by considering their vulnerabilities, can potentially reduce the risk of adopting new software systems.

Researchers in software reliability engineering have analyzed software defect finding rates. Software reliability growth models relate the number of defects found to the testing time (Lyu, 1995; Musa et al., 1987; Malaiya and Denton, 1997). Methods have been developed to project the mean time to failure (MTTF) or the failure rate that will occur after a specific period of testing. Software defect density (Malaiya and Denton, 2000; Mohagheghi et al., 2004; Mockus et al., 2002) has been a widely used metric to measure the quality of a program and is often used as a release criterion for a software project. Not much quantitative work, however, has been done to characterize security vulnerabilities along the same lines.

Security can be characterized by several possible metrics. Depending on how we view the systems, the applicable metrics can vary. Littlewood et al. (1993) and Brocklehurst et al. (1994) discuss some possible metrics to measure security based on dependability and reliability perspectives. They propose using effort rather than time to characterize the accumulation of vulnerabilities; however, they do not specify how to assess effort. Alves-Foss and Barbosa (1995) have proposed a metric termed Software Vulnerability Index (SVI). SVI is calculated using some predefined rules and can take values between 0 and 1.

An analysis of exploits for some specific vulnerabilities has been considered by Arbaugh and Fithen (2000) and Browne et al. (2001). The security intrusion process has also been examined by Jonsson and Olovsson (1997) and Madan et al. (2002). Other researchers have focused on modeling and designing tools that make some degree of security assessment possible (Schultz et al., 1990). Only a few studies have examined the number of vulnerabilities and their discovery rates. Rescorla (2005) has examined vulnerability discovery rates to determine the impact of vulnerability disclosures. Anderson (2002) has proposed a model for a vulnerability finding rate using a thermodynamics analogy. Alhazmi and co-workers (January 2005, August 2005, November 2005) have presented two models for the process of vulnerabilities discovery using data for Windows 98 and NT 4.0. In this work we focus on the density of defects in software that constitute vulnerabilities, using data from five versions of Windows and two versions of Red Hat Linux.

Vulnerability density is analogous to defect density. Vulnerability density may enable us to compare the maturity of the software and understand risks associated with its residual undiscovered vulnerabilities. We can presume that for systems that have been in deployment for a sufficient time, the vulnerabilities that have been discovered represent a major fraction of all vulnerabilities initially present. For relatively new systems, we would like to estimate the number of remaining vulnerabilities. This requires development and validation of appropriate vulnerability discovery models. Ounce Labs (2004) uses a metric termed V-density which appears to be somewhat related. However, their definition and evaluation approach is proprietary and is thus not very useful to the general community.

Unlike the data for general defects in a commercial operating system, which are usually hard to obtain, the actual data about known vulnerabilities found in major operating systems are available for analysis. We analyze this data to address a major question: do we observe any similarity in behavior for vulnerability discovery rates for various systems so that we can develop suitable models? We examine several software systems, which we grouped into related families after plotting the cumulative number of their vulnerabilities. One of our objectives is to identify possible reasons for the changes in the vulnerability detection trends.

One major difference makes interpreting the vulnerability discovery rate more difficult than the discovery rate of general defects in programs during testing. Throughout its lifetime after its release, an application program encounters changes in its usage environment. When a new version of a software system is released, its installed base starts to grow. As the newer version of the software grows, the number of installations of the older version starts to decline. The extent of vulnerability finding effort by both ''white hat'' and ''black hat'' individuals is influenced by the number of installations; this is because the larger the installed base the more is the reward for the effort. Thus, the rates at which the vulnerabilities are discovered are influenced by this variation in usage.

The rest of the paper is organized as follows. In Section 2 we introduce vulnerability density as a metric and examine its value for several major operating systems. We analyze, in Section 3, the vulnerability discovery rates. We consider two models for the vulnerability discovery process. We then examine in Section 4 the applicability of these models to several version of Windows as well as to Linux, an open-source operating system. Conclusions are presented in Section 5 and future research that is needed is identified.

2. Measuring systems' vulnerability density

We begin by introducing a new metric, *vulnerability density*, which describes one of the major aspects of security. Vulnerability density is a normalized measure, given by the number of vulnerabilities per unit of code size. Vulnerability density can be used to compare software systems within the same category (e.g., operating systems, web-servers, etc.).

To measure the code size we have two options. First, we can use the size of the installed system in bytes; the advantage of this measure is that information is readily available. However, this measure will vary from one installation to another. The second measure is the number of lines of source code. Here, we chose this measure for its simplicity and its correspondence to defect density metric in the software engineering domain. Let us now present a definition of vulnerability density (V_D):

Definition: vulnerability density is the number of vulnerabilities in the unit size of a code.

The vulnerability density is given by:

$$V_D = \frac{V}{S} \quad (1)$$

where S is the size of the software and V is the number of vulnerabilities in the system.

Following the common practice in software engineering, we consider 1000 source lines as the unit code size. When two systems, one large and one small, have the same defect density, they can be regarded as having similar maturity with respect to dependability. In the same manner, vulnerability density allows us to compare the quality of programming in terms of how secure the code is. If the instruction execution rates and other system attributes are the same, a system with a higher defect or vulnerability density is likely to be compromised more often.

Estimating the exact vulnerability density would require us to know the number of all the vulnerabilities of the system. Consequently, we define another measure in terms of the known vulnerabilities.

Definition: the known vulnerability density is the number of known vulnerabilities in the unit size of a code.

The *known vulnerability density* is given by:

$$V_{KD} = \frac{V_K}{S} \quad (2)$$

where V_K is the number of known vulnerabilities in the system. The *residual vulnerability density* (V_{RD}) is now defined as:

$$V_{RD} = V_D - V_{KD} \quad (3)$$

It is actually the residual vulnerability density (depending on vulnerabilities not yet discovered) that contributes to the risk of potential exploitation. Other aspects of the risk of exploitation include the time gap between the discovery of a vulnerability and the release and application of the corresponding patch. In this study we focus on vulnerabilities and their discovery. Recently there have been several attempts at comparisons between various attributes of open-source and commercial software (Mockus et al., 2002; Anderson, 2002). This is not, however, the focus of this paper. Rather, we want to probe the suitability of vulnerability density and vulnerability as metrics that can be used to assess and manage components of the security risk.

Vulnerability density can be used as a metric for estimating the number of residual vulnerabilities in a newly released software system given the size of the software. The vulnerability density of a newly released software system should be comparable to its comparable predecessor, given that it is designed using the same process. Thus, vulnerability density can be used by users to assess the risk when considering the purchase of a new software system. Developers can use the vulnerability density metric for planning for their testing process, and deciding when to stop testing. Testing is very expensive and delaying the release of a product can cause loss of market share. On the other hand, releasing a product with a potentially unacceptable number of vulnerabilities will cause customers to lose loyalty to the product. When the vendor has some objective assessment of the number of vulnerabilities existing in the system, the vendor can decide that the system is secure enough to be released.

Another application of the vulnerability density metric is maintenance planning. Integrating the use of vulnerability density with the vulnerability discovery models can allow estimation of the rate of discovery of vulnerabilities. This can enable the developer to get some objective estimation of numbers of vulnerabilities that are likely to be discovered within some forthcoming period of time. It can also be used by users and administrators to decide whether to delay application of the patches by examining the potential risks involved with such an exercise (Brykczynski and Small, 2003) so as to avoid destabilizing the software systems.

2.1. *Vulnerability density of the Windows family of operating systems*

For a metric to be useful, it should be predictable to some extent, if its values from similar projects are available. Here we examine the actual values of the vulnerability density metric

Table 1 – Vulnerability density versus defect density measured for some software systems

Systems	Msloc	Known defects	Known defect density (per Ksloc)	Known vulnerabilities	V_{KD} (per Ksloc)	V_{KD}/D_{KD} ratio (%)	Release date
Windows 95	15	5000	0.3333	50	0.0033	1.00	Aug 1995
Windows 98	18	10000	0.5556	84	0.0047	0.84	Jun 1998
Windows XP	40	106500	2.6625	125	0.0031	0.12	Oct 2001
Windows NT 4.0	16	10000	0.625	180	0.0113	1.80	Jul 1996
Win 2000	35	63000	1.80	204	0.0058	0.32	Feb 2000

Table 2 – Vulnerability density versus defect density measured for Red Hat Linux 6.2 and 7.1

Systems	Msloc	Known defects	Known defect density (per Ksloc)	Known vulnerabilities	V_{KD} (per Ksloc)	V_{KD}/D_{KD} ratio (%)	Release date
R H Linux 6.2	17	2096	0.12329	118	0.00694	5.63	Mar 2000
R H Linux 7.1	30	3779	0.12597	164	0.00547	4.34	Apr 2001

for some widely used systems. We also explore the relationship between the number of vulnerabilities and the total number of defects.

Table 1 presents values of the known defect density D_{KD} and known vulnerability density V_{KD} based on data from several sources (National Vulnerability Database, 2005; McGraw, 2003; Rodrigues, 2001; OS Data, 2004; MITRE Corporation, 2005) as of September 2005. Windows 95, Windows 98 and Windows XP are three successive versions of the popular Windows client operating system. We also include Windows NT and Windows 2000, which are successive versions of the Windows server operating systems. The known defect density values for Windows 95 and Windows 98 client operating systems are 0.33 and 0.55 per thousand lines of code, respectively. The higher defect density for Windows XP is due to the fact the data available are for the beta version. We can expect that the release version had significantly fewer defects. The defect density values for Windows NT and Windows 2000 are 0.6 and 1.8, respectively. The defects data for proprietary software systems are from published reports; for open-source systems the data are available from databases like Bugzilla (2005).

The known vulnerabilities column gives a recent count of the vulnerabilities found since the release date. We note that the vulnerability densities of Win 95 and Win 98 are quite close. The known vulnerability density for Win XP is 0.0020, much lower than the values for the two previous Windows versions. This is due to the fact that at this time V_{KD} represents only a fraction of the overall V_D. We can expect the number to go up significantly, perhaps to a value more comparable to the two previous versions as Win XP matures further.

We notice that the vulnerability density for Windows NT 4.0 is about three times that of Win 95 or Win 98. There are two possible reasons for this. Since Windows NT 4.0 is a server operating system, a larger fraction of its code involves external access, resulting in about three times the number of vulnerabilities. In addition, as a server operating system, it has possibly gone through more rigorous testing, resulting in the discovery of more vulnerabilities. Windows 2000 also demonstrates nearly as many vulnerabilities as NT, although due to its larger size, the vulnerability density is lower than that of NT.

One significant ratio to examine is V_{KD}/D_{KD}, which gives the fraction of defects that are vulnerabilities. Longstaff (2003) hypothetically assumes that vulnerabilities represent 5% of the total defects. Anderson (2002) assumes a value of 1% in his paper. Our results show that the values of the ratio are 1.00% and 0.84% for Win 95 and Win 98, respectively. For Windows XP, the number of known defects is given for the beta version, and is therefore higher than the actual number at release. In addition, since it was released last, relatively smaller fractions of XP vulnerabilities have thus far been found. This explains why the ratio of 0.12% for XP is significantly lower. We believe that this should not be used for comparison with other Windows versions. It is interesting to note that the ratio of 1% assumed by Anderson is within the range of values in Table 1.

Windows 2000 was an update of NT, with a significant amount of code added, much of which did not deal with external access; thus accounting for its relatively low ratio. In systems that have been in use for a sufficient time, V_{KD} is probably close to V_D. However, for newer systems we can expect that a significant number of vulnerabilities will be found in the near future. For a complete picture, we need to understand the process that governs the discovery of the remaining vulnerabilities. This is discussed in the next section.

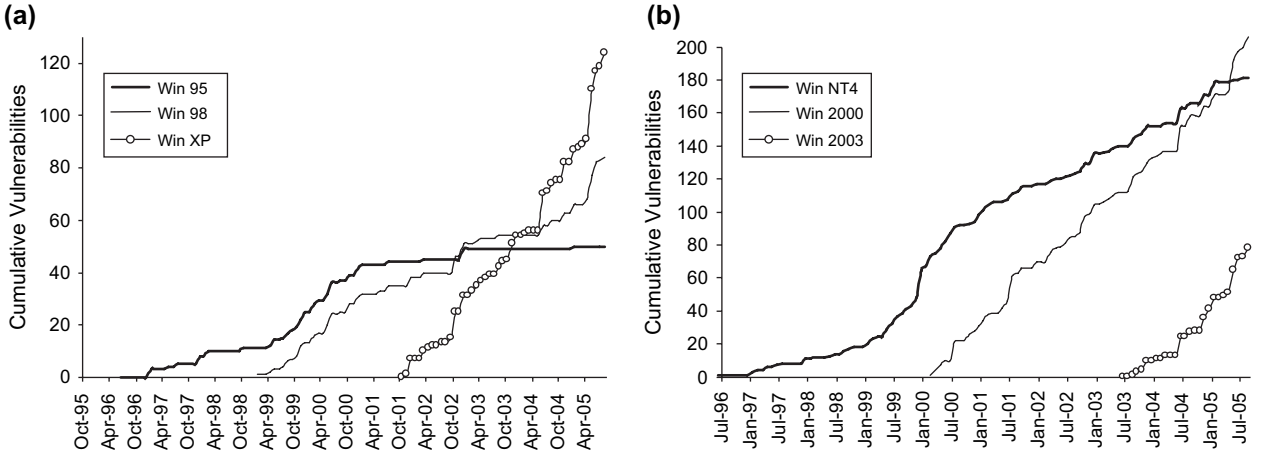

Fig. 1 – Cumulative vulnerabilities in Windows operating systems.

2.2. *Vulnerability density of the Red Hat Linux family of operating systems*

We next look at another popular group of operating systems —the Red Hat Linux family, which has had several versions. Linux is significantly different from the Windows family in that Linux is developed and maintained in a non-commercial open-source environment. We would like to see if vulnerability density and the ratio V_{KD}/D_{KD} are useful metrics and if the model of Eq. (5) applies for Linux.

It is not the discovered vulnerabilities but rather the vulnerabilities remaining undiscovered that form a significant component of the risk. In addition, the exploitation patterns and the timing of the patch releases also impact the risk. The V_{KD} value for Red Hat Linux 7.1 can be expected to rise significantly in near future, just as those of Windows XP. It is interesting to note that V_{KD}/D_{KD} ratio values for Linux are close to the value of 5% postulated by Longstaff (2003).

In Table 2 (MITRE Corporation, 2005; Bugzilla, 2005), we observe that although the code size for Linux 7.1 is twice as big as Linux 6.2, the defect density and vulnerability density values are remarkably similar. We also note that the V_{KD} values for the two versions of Red Hat Linux are significantly higher than for Windows 95 and Windows 98, and are approximately in the same range as for Windows 2000. However, V_{KD} alone should not be used to compare two competing operating system families.

3. Vulnerablity discovery rate

We now examine the rate at which the vulnerabilities are discovered. The vulnerability data is based on the data reported in National Vulnerability Database (2005) and MITRE Corporation (2005) as of February 2005. Software vulnerabilities reported by the Mitre Corporation has standardized the identifications of vulnerabilities. Candidate vulnerability is identified as CAN-XXXX; later, when the vulnerability is accepted the vulnerability identification number will be converted to a vulnerability entry with CVE-XXXX format. Both CVE and CAN have been used as equivalent vulnerabilities. Some vulnerabilities are reported before the release of a software system. Those vulnerabilities are found because of the shared components among consecutive versions, at about the same time of the release of the software. For consistency we have not included pre-release discovery of vulnerabilities.

3.1. *Observations on vulnerability discovery in operating systems*

Fig. 1 shows the vulnerability discovery rate for the five Windows family operating systems. Fig. 1(a) gives the cumulative vulnerabilities for Windows 95, Windows 98 and Windows XP (National Vulnerability Database, 2005). At the beginning, the curve for Windows 95 showed slow growth until about March 1998, after which it showed some saturation for several months. Windows 98 also showed relatively slow growth until about June 1998. After that, both Windows 95 and Windows 98 showed a faster rate of growth. The similarity of the plots in the later phase suggests that Windows 98 and Windows 95

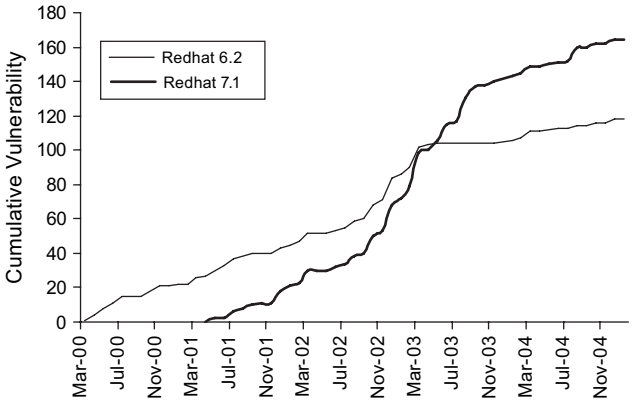

Fig. 2 – Cumulative vulnerabilities of Red Hat Linux versions 6.2 and 7.1.

shared a significant fraction of the code. The installed base of Windows 98 peaked during 1999–2000 (Alhazmi and Malaiya, January 2005). At some time after this, the discovery rates of vulnerabilities in both versions slowed down.

The saturation is more apparent in Windows 95. Based on our observation of shared vulnerabilities, we believe that many of the Windows 95 vulnerabilities discovered later were actually detected in the Windows 98 release. The cumulative vulnerabilities in Windows 95 and Windows 98 appear to have reached a plateau. Some vulnerabilities in Windows 98 were discovered rather late. This is explained by the code shared between the Windows 98 and Windows XP versions, as discussed next.

Fig. 1 demonstrates that Windows XP showed swift growth in vulnerabilities with respect to its release date. There were also many vulnerabilities shared with Windows 98. However, XP has its own unique vulnerabilities, and they form the majority. Windows XP shows practically no learning phase; rather, the plot shows a linear accumulation of vulnerabilities. The slope is significantly sharper than for Windows 98. The sharpness of the curve for XP is explained by its fast adoption rate (Alhazmi and Malaiya, January 2005), making finding vulnerabilities in XP more rewarding. Windows 98 has showed

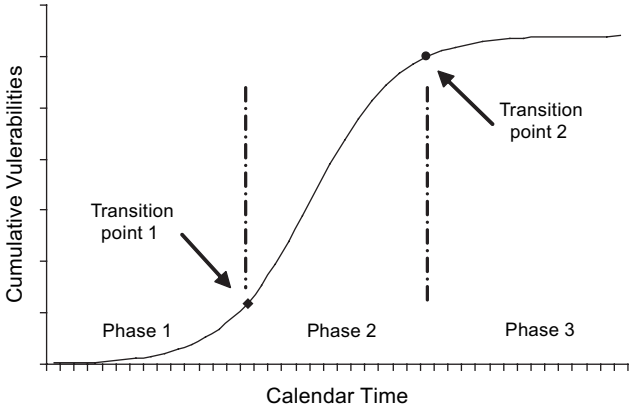

Fig. 3 – The three-phases of the vulnerability discovery process.

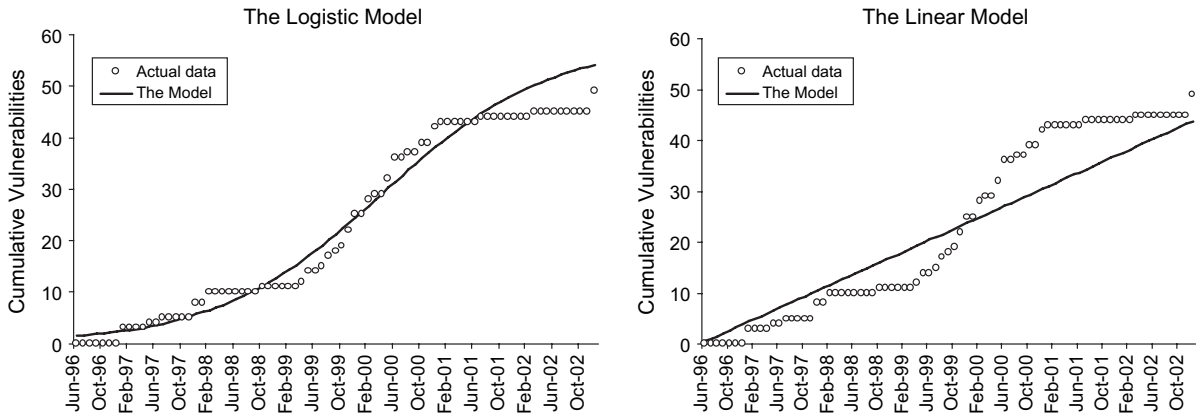

Fig. 4 – Windows 95 data fitted to the model.

a longer learning phase followed by a linear accumulation, later followed by saturation. Consequently, it appears that Windows XP has contributed to the detection of most of the later Windows 98 vulnerabilities.

The data for the three operating systems demonstrate that there is significant interdependence among vulnerability discovery rates for the three versions. This interdependence is due to the sharing of codes. The shifting shares of the installed base need to be taken into account when examining the vulnerability discovery trends. Windows 98 represents a middle stage between the other two versions from the perspective of vulnerability detection.

Fig. 1(b) shows the vulnerabilities in Windows NT and Windows 2000 and some initial data for Windows 2003. We observe that that both Windows NT and Windows 2000 gained installed base gradually (Alhazmi and Malaiya, January 2005), and users did not switch over to from Windows NT to Windows 2000 quickly. The use of Windows NT peaked around end of 2001, but its share did not drop dramatically as the share for Win 2000 grew. The figure also shows, a later system Windows 2003 Server, it has close relationship with Windows 2000 which made the learning phase shorter.

Fig. 2 shows two versions of Red Hat Linux, versions 6.2 and 7.1. In both, we observe saturation in the later period.

3.2. *Modeling the vulnerability discovery process*

From the data plotted in Figs. 1 and 2, we can see a common pattern of three phases in the cumulative vulnerabilities plot of a specific version of an operating system, as shown in Fig. 3, except for Windows 2000, which show a rather linear trend.

During these three phases, the usage environment changes, thereby impacting the vulnerability detection effort. In Phase 1, the operating system starts attracting attention and users start switching to it. The software testers (including hackers and crackers) begin to understand the target system and gather sufficient knowledge about the system to break into it successfully. In Phase 2, the acceptance of the new system starts gathering momentum. It continues to increase until the operating system reaches the peak of its popularity. This is the period during which discovering its vulnerabilities will be most rewarding for both white hat and black hat finders. After a while, in Phase 3, the system starts to be replaced by a newer release. The vulnerability detection effort will then start shifting to the new version. The technical support for that version and hence the frequency of update patches will then begin to decline.

This S-shaped behavior shown in Fig. 3 can be described by a time-based model logistic proposed by Alhazmi and Malaiya

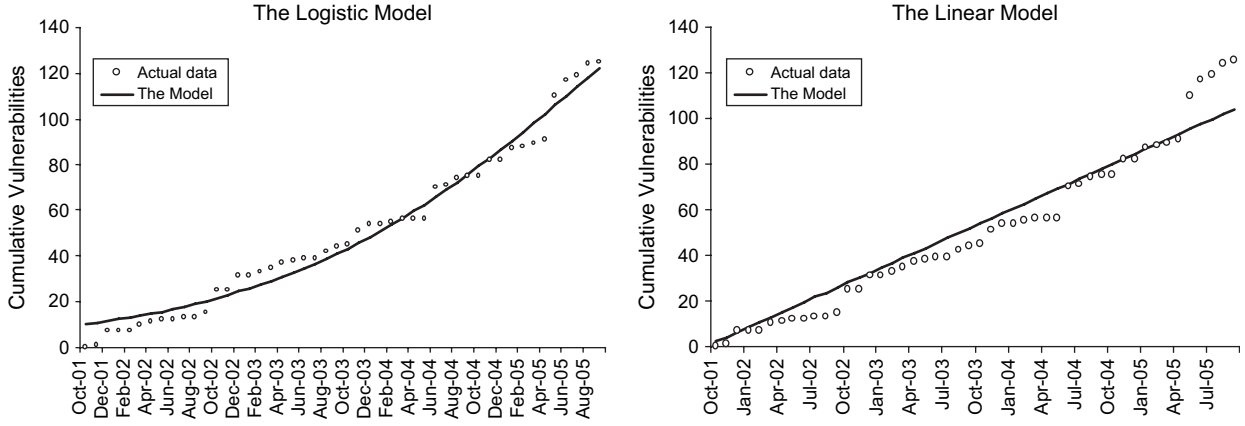

Fig. 5 – Windows XP data fitted to the model.

Table 3 – χ^2 goodness of fit test results for the logistic model

System	A	B	C	DF	χ^2	$\chi^2_{critial}$ (5%)	P-value	Result
Windows 95	0.001691	50.80229	0.8064509	79	47.71	100.75	0.999991	Significant
Windows 98	0.000944	69.94451	0.136279	81	90.84	103.01	0.213	Significant
Windows XP	0.000279	241.3255	0.101528	48	52.10	65.17	0.317	Significant
Windows NT 4.0	0.000584	153.62	0.47	103	83.29	127.69	0.923	Significant
Windows 2000	0.00036	197.7525	0.060362	68	87.78	88.25	0.054	Significant

(January 2005). Let y be the cumulative number of vulnerabilities. We assume that the vulnerability discovery rate is controlled by two factors. The first of these is due to the momentum gained by the market acceptance of the product; this is given by a factor Ay, where A is a constant of proportionality. The second factor is saturation due to a finite number of vulnerabilities and is proportional to $(B - y)$, where B is the total number of vulnerabilities. The vulnerability discovery rate is then given by the following differential equation,

$$\frac{dy}{dt} = Ay(B - y) \tag{4}$$

where t is the calendar time. By solving the differential equation, we obtain

$$y = \frac{B}{BCe^{-ABt} + 1} \tag{5}$$

where C is a constant introduced while solving Eq. (4). It is thus a three-parameter model. In Eq. (5), as t approaches infinity, y approaches B, as the model assumes. The constants A, B and C need to be determined empirically using the recorded data.

By mathematically analyzing the model we can characterize some useful information. For example, we can determine what the maximum vulnerability discovery rate is. By taking the first derivative we can see how the rate changes. Hence, by taking the first derivative of the equation given in Eq. (5), the rate of vulnerability discovery is:

$$\frac{d\Omega}{dt} = \frac{AB^3Ce^{-ABt}}{(BCe^{-ABt} + 1)^2} \tag{6}$$

From Eq. (6), the highest vulnerability discovery rate occurs at the midpoint at:

$$T_m = -\ln\left[\frac{1}{BC}\right]/AB \tag{7}$$

By substituting Eq. (7) in (6) we get the maximum rate of vulnerability discovery which equals $AB^2/4$.

The authors recently proposed an alternative effort-based model (Alhazmi and Malaiya, January 2005), which also fits well; however, it requires extensive usage data collection. A time-based model was derived by Anderson (2002); however, its applicability to actual data has not yet been studied.

From Fig. 1, we observe that Windows XP and Windows 2000 datasets show a linear behavior. In these cases, we can assume that the trend corresponds to the linear part of the curve corresponding to Phase 2, between the two transition points. Here we test the applicability of the linear model which assumes a constant vulnerability discovery rate. The linear model can be seen as an approximation to the S-shaped model. The linear vulnerability model is as follows:

$$\Omega(t) = S \times t + k \tag{8}$$

where S denotes the slope (i.e. vulnerability discover rate); and k is a constant.

Because of its simplicity, the linear model will be easier to use. In cases where saturation has not been reached, it may fit reasonably well. It cannot be expected to work over a very long time span, because it will predict infinite number of vulnerability if the calendar time approaches infinity.

4. Examining goodness of fit for the two models

A goodness of fit analysis is now performed using chi-square with $\alpha = 5\%$. We compare the actual month-by-month data with the fitted model. The null hypothesis is that the model fits the data. It will be rejected when $(\chi^2 > \chi^2_{critical})$ indicating a bad fit. The chi-square goodness of fit test given by:

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

where o_i and e_i are the observed and the expected values, respectively. With the significance level at 5%, let us consider the following:

H_0: the model fits the data (when $\chi^2 \le \chi^2_{critical}$).
H_1: the model does not fit the data (when $\chi^2 > \chi^2_{critical}$).

Table 4 – χ^2 goodness of fit test results for the linear model

System	Slope (s)	k	DF	χ^2	$\chi^2_{critial}$ (5%)	P-value	Result
Windows 95	0.599367	0	79	128.80	100.75	0.0034	Not significant
Windows 98	0.850455	4.040356	81	55.00	103.01	0.9519	Significant fit
Windows XP	2.0949	0	48	62.36	65.17	0.0797	Significant fit
Windows NT 4.0	1.669254	0	103	456.22	127.69	0	Not significant
Windows 2000	2.745798	1.527603	68	30.05	88.25	0.9999	Significant

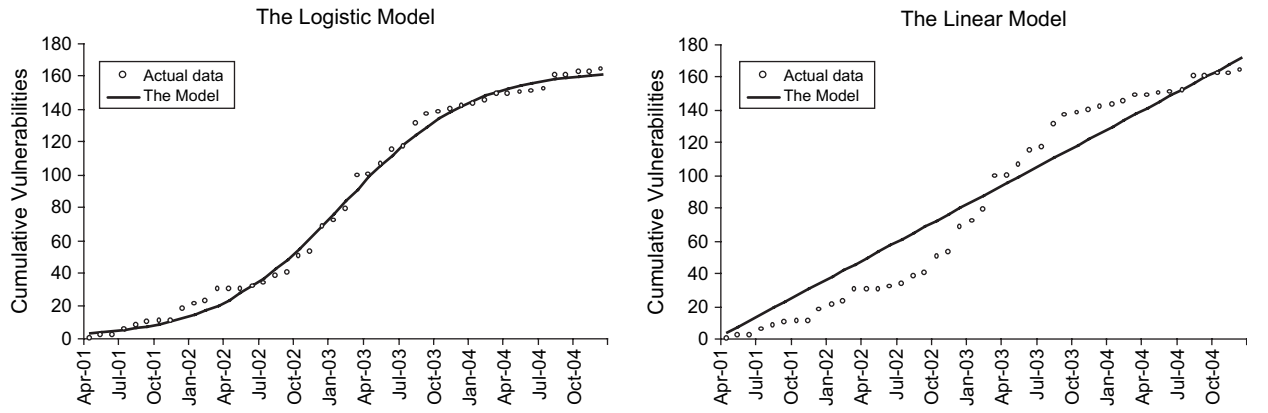

Fig. 6 – Red Hat Linux 7.1 fitted to the model.

4.1. *Goodness of fit for the Windows family*

Figs. 4 and 5 give the datasets of Windows 95 and Windows XP fitted to the logistic and the linear models, respectively. The numerically obtained model parameters are given in Tables 3 and 4.

The null hypotheses was accepted using the logistic model, for all the datasets in Table 3, because $\chi^2 \leq \chi^2_{\text{critical}}$ for all cases as indicated by a chi-squared value less than the critical value at the 95% significance level. That is, the fit is found to be statistically significant,

By analyzing the data in Table 4, we observe that the linear model performs well when the saturation phase has not yet been reached, as seen for both Windows 2000 and Windows XP. The trend suggests that the finders have not lost their interest in finding new vulnerabilities in them. Another factor is when the vulnerability dataset contains a significant number of shared vulnerability with a later system; in this case, the data appeared to be a super imposition of two or more consecutive S-shaped models. This is noticeable in Windows 98 where most of the late discovered vulnerabilities were also reported in Windows XP. Some systems may have a shorter learning phase, due to some easy vulnerabilities occurring early after the release or when there is close relationship with existing software, as we observe for Windows XP and Windows 2000.

The S-shaped model has fitted all the datasets, because it incorporates the learning and saturating phases. It can also fit linear data reasonably well. More matured systems should eventually exhibit saturation resulting in a better fit with the logistic model. Note that in many cases sudden acceleration in the popularity of a product increases the risks significantly as more attention is paid to the testing process.

We have statistically examined the linear model on the same sets of operating systems; the fit was significant in some and non-significant in others. Table 4 and Table 6 detail the chi-square test results. We have not constrained the slope; only k has been constrained to be positive, this is a reasonable assumption, since the number of vulnerabilities cannot be negative at any time.

4.2. *Goodness of fit of the Linux operating system*

Fig. 6 presents the datasets for Red Hat Linux 7.1, fitted to both linear and logistic models. The model parameter values and the results of the chi-square test are given in Tables 5 and 6. Again, the application of the chi-squared test shows that the fit is significant in the case of the logistic model. However, Red Hat 7.1 dataset did not fit the linear model.

We would like to be able to project the expected number of vulnerabilities that will be found during the major part of the lifetime of a release, using early data and a model like the one given in Eq. (5). This would require an understanding of the three parameters involved and developing methods for robust estimation.

5. Conclusions

In this paper, we have explored the applicability of quantitative metrics describing vulnerabilities and the process that governs their discovery. We have examined the data for five of the most widely used operating systems, including three successive version of Windows and two versions of Red Hat Linux. We have evaluated the known *vulnerability densities* in the five operating systems. The lower value for Win XP relative to Win 95 and Win 98 is attributable to the fact that a significant fraction of Win XP vulnerabilities have not yet been discovered.

Table 5 – χ^2 goodness of fit test results for the logistic model

Systems	A	B	C	DF	χ^2	χ^2_{critical} (5%)	P-value	Result
Red Hat Linux 6.2	0.000829	123.94	0.12968	58	34.62	76.78	0.99997	Significant
Red Hat Linux 7.1	0.001106	163.9996	0.37999	45	27.63	61.66	0.989	Significant

Table 6 – $\chi 2$ goodness of fit test results for the linear model

System	Slope (s)	k	DF	χ^2	$\chi^2_{critical}$ (5%)	P-value	Result
Red Hat Linux 6.2	2.227409	0	58	43.06	76.78	0.928519784	Significant
Red Hat Linux 7.1	3.819908	0	45	376.98	61.66	1.08875E-20	Not significant

We observe that the values of vulnerability densities fall within a range, and for similar products they are closer together. This is same case with software defect densities. We note that the ratio of vulnerabilities to the total number of defects is often in the range of 1–5%, as was speculated to be the case by some researchers. As we would expect, this ratio is often higher for operating systems intended to be servers. The results indicate that vulnerability density is a significant and useful metric. We can expect to gain further insight into vulnerability densities when additional data, together with suitable quantitative models, are available. Such models may allow empirical estimation of vulnerability densities along the lines of similar models for software cost estimation or software defect density estimation.

This paper presented graphs showing the cumulative number of vulnerabilities for the five operating systems. The vulnerabilities shared by successive versions are also given. These plots are analogous to *reliability growth* plots in software reliability. However, there are some significant differences. The initial growth rate at the release time is small but subsequently accelerates. Generally the plots show a linear trend for a significant period. These plots tend to show some saturation, often followed by abrupt increases later. This behavior is explained by the variability of the effort that goes into discovering the vulnerabilities. The models given by Eqs. (5) and (8) are fitted to vulnerability data for the seven operating systems. For the logistic model the fit is found to be statistically significant in all cases. The linear model does a reasonable job in a few cases, especially where no saturation has yet set in.

The code shared by a new and hence a competing version of the operating system can impact the vulnerability discovery rate in a previous version. Further research is needed to model the impact of the shared code. We expect that with further research and significant data collection and analysis, it will be possible to develop reliable quantitative methods for security akin to those used in the software and hardware reliability fields.

REFERENCES

Alhazmi OH, Malaiya YK. Quantitative vulnerability assessment of systems software. In: Proceedings of 51st annual reliability and maintainability symposium, Alexandria, VA; January 2005. p. 615–20.

Alhazmi OH, Malaiya YK. Modeling the vulnerability discovery process. In: International Symposium on Software Reliability Engineering; November 2005.

Alhazmi OH, Malaiya YK, Ray I. Security vulnerabilities in software systems: a quantitative perspective. In: Proceedings of IFIP WG 11.3 working conference on data and applications security; August 2005. p. 281–94.

Alves-Foss Jim, Barbosa Salvador. Assessing computer security vulnerability. Operating Systems Review 1995;29(3):3–13.

Anderson Ross. Security in open versus closed systems—the dance of Boltzmann, Coase and Moore. In: Conference on open source software: economics, law and policy, Toulouse, France; June 2002. p. 1–15, http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/toulouse.pdf.

Arbaugh WA, Fithen WL, McHugh J. Windows of vulnerability: a case study analysis. IEEE Computer December 2000;33(12):52–9.

Brocklehurst S, Littlewood B, Olovsson T, Jonsson E. On measurement of operational security. In: Proceedings of the 9th annual IEEE conference on computer assurance. Gaithersburg: IEEE Computer Society; 1994. p. 257–66.

Browne HK, Arbaugh WA, McHugh J, Fithen WL. A trend analysis of exploitation. In: Proceedings of the IEEE Symposium on Security and Privacy; May 2001. p. 214–29.

Brykczynski B, Small RA. Reducing internet-based intrusions: effective security patch management. IEEE Software January–February 2003;20(1):50–7.

Handbook of software reliability engineering. In: Lyu MR, editor. McGraw-Hill; 1995.

Jonsson E, Olovsson T. A quantitative model of the security intrusion process based on attacker behavior. IEEE Transactions on Software Engineering April 1997;24(3):235–45.

Littlewood B, Brocklehurst S, Fenton N, Mellor P, Page S, Wright D. Towards operational measures of computer security. Journal of Computer Security 1993;2(2/3):211–30.

Longstaff T. CERT experience with security problems in software. Carnegie Mellon University, http://research.microsoft.com/projects/SWSecInstitute/slides/Longstaff.pdf; June 2003.

Madan BB, Goseva-Popstojanova K, Vaidyanathan K, Trivedi KS. Modeling and quantification of security attributes of software systems. In: Proceedings of the IEEE international performance and dependability symposium (IPDS 2002); June 2002.

Malaiya YK, Denton J. What do the software reliability growth model parameters represent? In: International symposium on software reliability engineering; 1997. p. 124–35.

Malaiya YK, Denton J. Module size distribution and defect density. In: Proceedings of the IEEE international symposium on software reliability engineering; October 2000. p. 62–71.

McGraw G. From the ground up: the DIMACS software security workshop. IEEE Security & Privacy March/April 2003;1(2):59–66.

Mockus A, Fielding RT, Herbsleb J. Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology 2002;11(3):309–46.

Mohagheghi P, Conradi R, Killi OM, Schwarz H. An empirical study of software reuse vs. defect-density. In: Proceedings of the 26th international conference on software engineering; May 2004. p. 282–91.

Musa JD, Ianino A, Okumuto K. Software reliability measurement prediction application. McGraw-Hill; 1987.

National Vulnerability Database; February 2005. <http://nvd.nist.gov>.

OS Data, Windows 98, <http://www.osdata.com/oses/win98.htm>; March 2004.

Ounce Labs. Security by the numbers: the need for metrics in application security, <http://www.ouncelabs.com/library.asp>; 2004.

Pfleeger Charles P. Security in computing. Prentice-Hall; 1997.
Red Hat Bugzilla, <https://bugzilla.redhat.com/bugzilla>; January 2005.
Rescorla E. Is finding security holes a good idea? Economics of Information Security January–February 2005;3(1):14–9.
Rodrigues P. Windows XP Beta 02. Only 106,500 bugs, <http://www.lowendmac.com/tf/010401pf.html>; August 2001.
Schultz Jr EE, Brown DS, Longstaff TA. Responding to computer security incidents. Lawrence Livermore National Laboratory, ftp://ftp.cert.dfn.de/pub/docs/csir/ihg.ps.gz; July 23, 1990.
The MITRE Corporation, <http://www.mitre.org>; February 2005.

Omar H. Alhazmi is a Ph.D. student of Computer Science at Colorado State University which he joined in January 2002. He will finish his Ph.D. in 2006. He received his Master's degree in Computer Science from Villanova University in 2001. Mr. Alhazmi's interests are in the areas of software engineering and computer security.

Yashwant K. Malaiya is a Professor in the Computer Science Department at Colorado State University. He received his M.S. in Physics from Sagar University, MScTech in Electronic from Birla Institute of Technology, Pilani and Ph.D. in Electrical Engineering from Utah State University. He has published widely in the areas of fault modeling, software and hardware reliability, testing and testable design since 1979. He has also been a consultant to industry. He was the General Chair of 1993 and 2003 IEEE International Symposium on Software Reliability Engineering (ISSRE).

Indrajit Ray is an Assistant Professor in the Computer Science Department at Colorado State University. He received his M.E. in Computer Science and Engineering from Jadavpur University and Ph.D. from George Mason University. He has published widely in the areas of security models, security protocols, e-commerce protocols, computer forensics, database security and network security since 1997. He is one of the founding members and current chair of the IFIP TC-11 Working Group 11.9 on Digital Forensics.