Security Vulnerabilities in Software Systems: A Quantitative Perspective

Omar Alhazmi, Yashwant Malaiya, and Indrajit Ray

Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA
{omar, malaiya, indrajit}@cs.colostate.edu

Abstract. Security and reliability are important attributes of complex software systems. It is now common to use quantitative methods for evaluating and managing reliability. 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 identified in a future release of a software system. We use several major operating systems as representatives of complex software systems. The data on vulnerabilities discovered in some of the popular operating systems is analyzed. We examine this data to determine if the density of vulnerabilities in a program is a useful measure. We try to identify 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 vulnerability discovery may be influenced by several factors including sharing of codes between successive versions of a software system.

1 Introduction

Reliance on networked systems has brought the security of software systems under considerable scrutiny. 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 risks to be evaluated quantitatively. Quantitative methods can permit resource allocation for achieving a desired security level, as it is done for software or system reliability. Thus far, only limited attention has been paid 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 problem of quantifying vulnerabilities in a complex software system. Security vulnerabilities are " defect(s) which enables an attacker to bypass security measures" [1]. Malicious attackers seek to identify and exploit system vulnerabilities to cause security breaches. Reducing the number of vulnerabilities in a system is thus of utmost importance.

S. Jajodia and D. Wijesekera (Eds.): Data and Applications Security 2005, LNCS 3654, pp. 281–294, 2005.

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 defect, a similar measure for estimating security vulnerabilities appears long overdue. In this paper, we quantitatively examine the number of vulnerabilities in several popular operating systems. Such quantitative characterization of vulnerabilities can be used to evaluate metrics that can guide the allocation of resources for security testing, development of security patches and scheduling their releases. It can also be used by end-users to assess risks and estimate the needed redundancy in resources and procedures for handling potential security breaches.

It is not possible to guarantee absence of defects in non-trivial 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 [2]. Nonetheless, examination of defect densities (that is the number of defects identified in the unit size of the software code) is still 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 [3]. 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 [2,3,4]. 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 [5,6,7] 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. Very little quantitative work has been done to characterize security vulnerabilities along the same lines.

Security can be characterized by several possible metrics. Littlewood et al. [8,9] 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. An analysis of exploits for some specific vulnerabilities has been considered by Arbaugh [10] and Browne [11]. The security intrusion process has also been examined by Johnson and Olovsson [12] and Madan et al. [13]. Other researchers have focused on modeling and designing tools that make some security assessment possible [1]. Only a few studies have examined the number of vulnerabilities and their discovery rates. Rescorla [14] has examined vulnerability discovery rates to determine the impact of vulnerability disclosures. Anderson [15] has proposed a model for a vulnerability-finding rate using a thermodynamics analogy. Alhazmi and Malaiya [16] have presented two models for the process of vulnerabilities discovery using data for Windows 98 and NT 4.0. In the current 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 uses a metric termed V-density [17] 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 group into related families after determining 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 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 the major terms we use in this work. We analyze, in section 3, the data for some of the Windows operating systems to evaluate the densities of vulnerabilities that are known. We talk about the remaining vulnerabilities (yet to be discovered) in section 3.1. In section 3.2 we present a model for the vulnerability discovery process. We then examine in section 4 the applicability of our major observations for two version of Linux, an open-source operating system. Finally, we conclude the paper in section 5 by identifying future research that is needed.

2 Measuring Systems' Vulnerability Density

We begin by introducing a new metric, vulnerability density, that 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, webservers, 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 source lines of code. Here, we chose this mea-

sure for its simplicity and its correspondence to defect density metric in the software engineering domain. Let us now present a definition of vulnerability density (VD):

Definition 1. *Vulnerability density is the number of vulnerabilities in the unit size of a code. It is given by*

$$V_d = \frac{V}{S} \tag{1}$$

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 one thousand 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 2. *The known vulnerability density is the number of known vulnerabilities in the unit size of a code. The known vulnerability density is given by*

$$V_{kd} = \frac{V_k}{S} \tag{2}$$

where is the number of known vulnerabilities in the system.

It is the residual vulnerability density (VRD) given by

$$V_{rd} = V_d - V_{kd} \tag{3}$$

that (depending on vulnerabilities not yet discovered) 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 a patch. In this study we focus on vulnerabilities and their discovery. Recently there have been a number of comparisons between several attributes of open-source and commercial software [7,15]. 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.

3 The Windows Family of Operating Systems

Table 1 presents values of the known defect density D_{KD} and known vulnerability density V_{KD} based on data from several sources [18,19,20,21] as of January 2005. Windows 95, 98 and 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

Table 1. Vulnerability density vs. defect density measured for some software systems

Systems	*Msloc*	*Known Defects*	*Known Defect Density* (per Ksloc)	*Known Vulnera-bilities*	V_{KD} (per Ksloc)	V_{KD} / D_{KD} *Ratio* (%)	*Release Date*
Windows 95	15	5	0.3333	50	.0033	1.00%	Aug 1995
Windows 98	18	10	0.5556	66	.0037	0.66%	Jun 1998
Windows XP	40	106.5	2.6625	88	.0022	0.08%	Oct 2001
Windows NT 4.0	16	10	0.625	179	.0112	1.79%	Jul 1996
Windows 2000	35	63	1.80	170	.0049	0.27%	Feb 2000

lines of code, respectively. The higher defect density for Windows XP is due to the fact the data available is for the beta version. We can expect that the release version had significantly fewer defects. The defect density values for Windows NT and 2000 are 0.6 and 1.8, respectively. 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 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 VD. We can expect the number to go up significantly, perhaps to a value more comparable to the two previous versions. 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 NT is a server, a larger fraction of its code involved external access, resulting in about three times the number of vulnerabilities. In addition, as a server operating system, it must have gone through more thorough 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, which gives the fraction of defects that are vulnerabilities. McGraw [19] hypothetically assumed that vulnerabilities might represent 5% of the total defects. Anderson assumed a value of 1% in [15]. Our results show that the values of the ratio are 1.00% and 0.66% for Win95 and Win98. 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, smaller fractions of XP vulnerabilities have thus far been found. This explains why the ratio of 0.083% for XP is significantly lower. We believe that this should not be used in a 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, VKD is probably close to VD. 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, as discussed in the next sub-section.

3.1 An Examination of the Remaining Vulnerabilities

We now examine the rate at which vulnerabilities were reported in the five operating systems, as shown in Figures 1-3. Some specific vulnerabilities are shared by successive versions of the system. Such shared vulnerabilities are shown using a separate plot. The data show that some vulnerabilities were reported even before the general release date of a particular version. For consistency, we omit vulnerabilities encountered before the release date of a particular version. Figure 1 gives the cumulative vulnerabilities for Windows 95 and 98 [18]. 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 shared a significant fraction of the code. The installed base of Windows 98 peaked during 1999-2000 [16]. 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 98 and XP versions, as discussed next. Figure 2 gives the cumulative vulnerabilities in Windows 98 and XP [18]. It 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

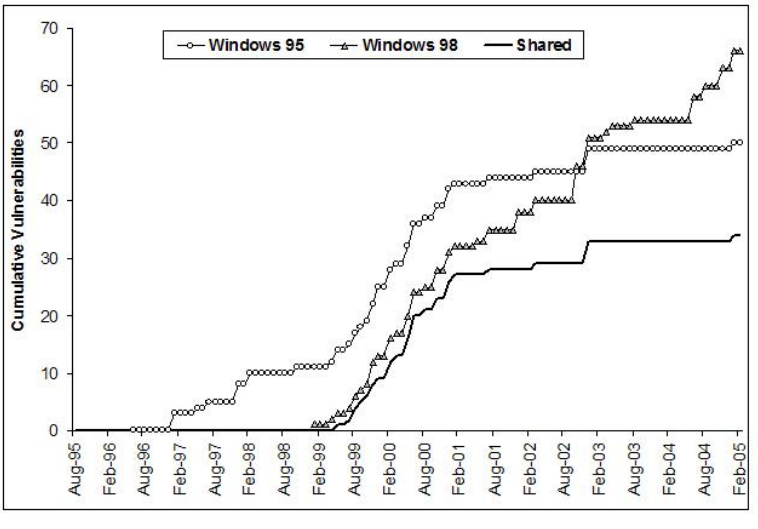

Fig. 1. Cumulative vulnerabilities in Windows 95 and Windows 98

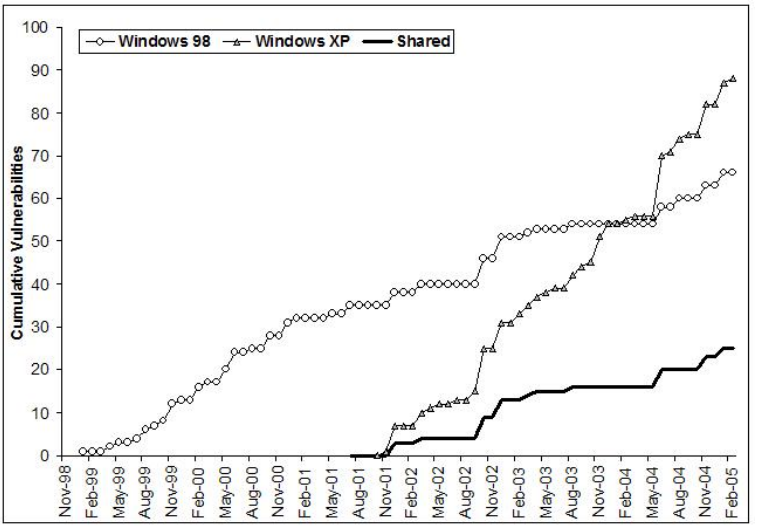

Fig. 2. Cumulative vulnerabilities in Windows 98 and its successor Windows XP

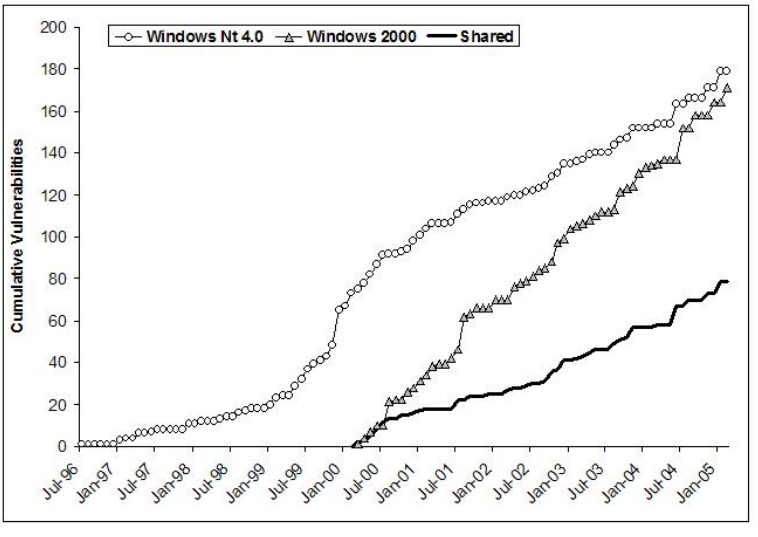

Fig. 3. Cumulative vulnerabilities of Windows NT and Windows 2000

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 [16], making finding vulnerabilities in XP more rewarding. Windows 98 has showed a longer learning phase followed by a linear accumulation, later followed by saturation.

The relationship between the vulnerabilities reported in the older Windows 95, Windows 98 and Windows XP is important. As we can observe in Figure 1, Windows 98 inherited most of the earlier vulnerabilities found in Windows 95. Vulnerabilities reported for Windows 98 slowed down at some point, only to pick up again when Windows XP was released. 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 demonstrates 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. Figure 3 shows the vulnerabilities in Windows NT and 2000; the shared vulnerabilities are also shown. Unlike the two previous figures, we do not observe a prominent time-lag between the two. The reason is that both of them gained installed base gradually [16]. The use of NT peaked around end of 2001, but its share did not drop dramatically as the share for Win2000 grew.

3.2 Modeling the Vulnerability Discovery Process

From the data plotted in the figures above, 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 Figure 4.

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

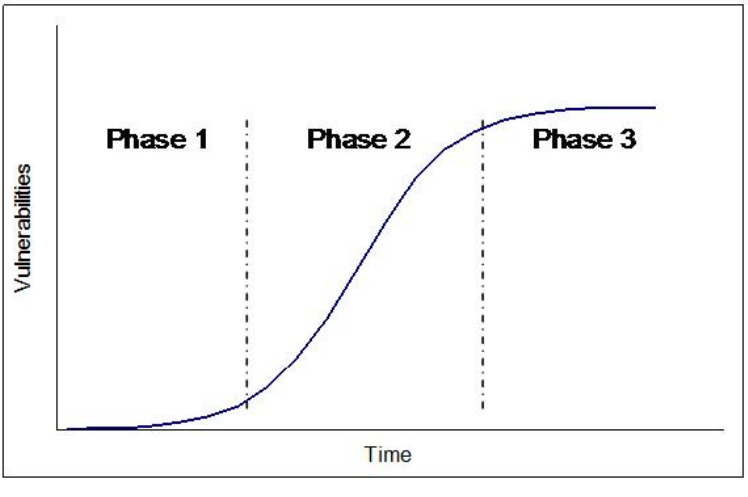

Fig. 4. The 3-phases of the vulnerability discovery process

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 Figure 4 can be described by a time-based model introduced earlier by Alhazmi and Malaiya [16]. 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 Equation 4. It is thus a three-parameter model. In Equation 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. An alternative effort-based model [16], which also fits well but requires extensive usage data collection was recently proposed by the authors. A time-based model was derived by Anderson [15]; however, its applicability to actual data has not yet been studied.

Figures 5 and 6 give the data for Windows 95 and NT 4.0 with a fitted plot according to the model given in Equation 5. The numerically obtained model parameters are given in Table 2. We have used chi-squared goodness of fit test to evaluate the applicability of the model. The fit is found be statistically significant, as indicated by a chi-squared value less than the critical value at the 95% significance level. A similar analysis for Windows 98, XP and 2000 also demonstrate a good fit of the model to the data.

Table 2. χ^2 goodness of fit test results

Systems	*A*	*B*	*C*	χ^2	$\chi^2_{critical}$ (5%)	*P-value*
Windows 95	0.001938	49.5	1.170154	40.72	119.87	0.9999998
Windows 98	0.001049031	66	0.140462	64.79	96.2	0.742
Windows XP	0.001391	88	0.190847	25.75	56.94	0.961
Windows NT 4.0	0.000584	153.62	0.47	82.3942	127.69	0.923
Windows 2000	0.000528	163.96	0.073187	60.91	80.23	0.444

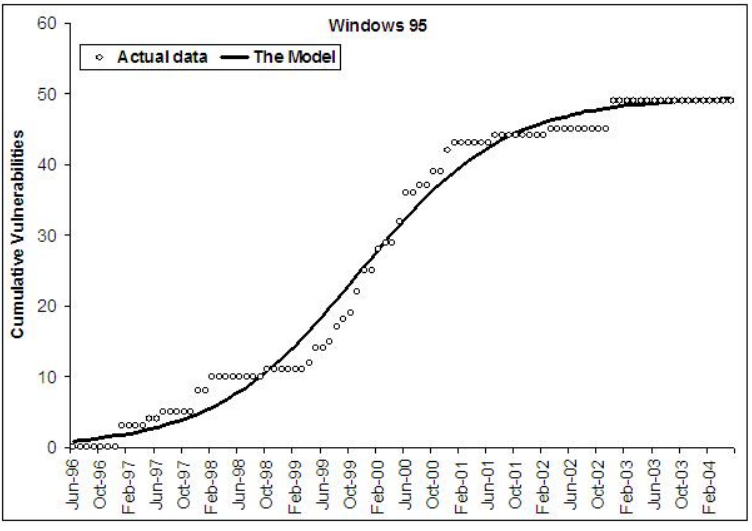

Fig. 5. Windows 95 data fitted to the model

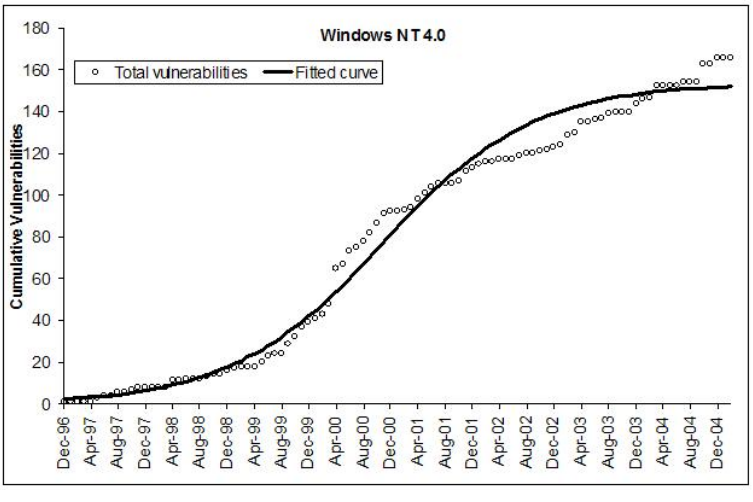

Fig. 6. Windows NT 4.0 data fitted to the model

4 Linux Operating System

We examine two versions of Red Hat Linux, versions 6.2 and 7.1, shown in Figure 7. In both, we observe saturation in the later period. We note that in the later duration, a majority of the vulnerabilities discovered in version 6.2 are in fact shared.

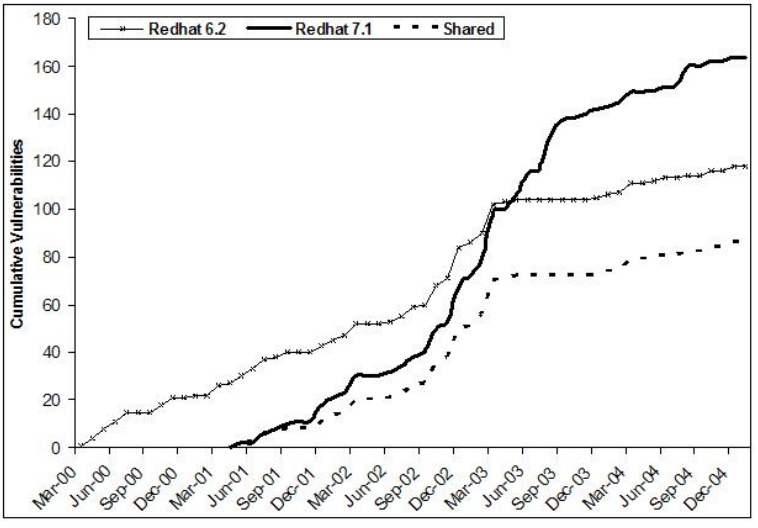

Fig. 7. Cumulative vulnerabilities of Red Hat Linux version 6.2 and 7.1

Table 3. Vulnerability density vs. defect density measured for Red Hat Linux 6.2 and 7.1

Systems	*Msloc*	*Known Defects*	*Known Defect Density* (per Ksloc)	*Known Vulnera-bilities*	V_{KD} (per Ksloc)	V_{KD}/ D_{KD} *Ratio* (%)	*Release Date*
R H Linux 6.2	17	2096	0.12329	118	.00694	5.63%	Mar 2000
R H Linux 7.1	30	3779	0.12597	164	.00547	4.34%	Apr 2001

In Table 3 [22,23], 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 note that the VKD values for the two versions of Red Hat Linux are significantly higher than for Windows 95 and 98, and are approximately in the same range as for Windows 2000. However, VKD alone should not be used to compare of the two competing operating system families. 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 VKD 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 ratio values for Linux are close to the value of 5% postulated by McGraw [19].

Figure 8 presents the raw data for 7.1, together with the fitted model. The model parameter values and the results of the chi-squared test are given in Table 4. Again, the application of the chi-squared test shows that the fit is significant.

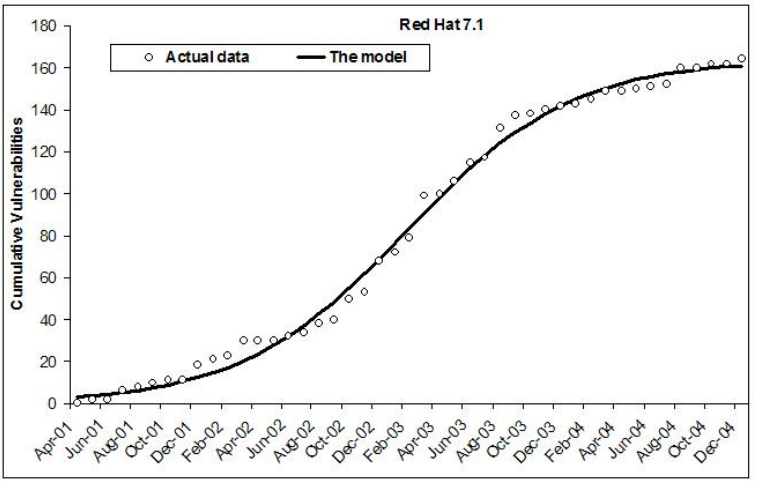

Fig. 8. Red Hat Linux 7.1 fitted to the model

Table 4. χ^2 goodness of fit test results

Systems	*A*	*B*	*C*	χ^2	$\chi^2_{critical}$ (5%)	*P-value*
Red Hat Linux 6.2	0.000829	123.9393	0.129678	34.62	76.78	0.999974
Red Hat Linux 7.1	0.001106	163.9996	0.379986	27.62715	61.65623	0.989

While the model of Equation 5 fits the data for all the operating systems examined here, some aspects of the process need further examination. There is often a significant overlap between two successive operating systems. The attention received by a version, n, results in detection of vulnerabilities not only in version n but also in the code shared between versions n and (n-1). This, in turn, results in a bump-up in the version (n-1) discovery rate, even though its installed base may be shrinking rapidly. This overlap causes some deviation for the model. Techniques need to be developed for modeling this overlap in order to achieve higher accuracy. 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 Equation 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 98 is attributable to the fact that a significant fraction of Win XP vulnerabilities have not yet been discovered. As has been observed for software defect densities, the values of vulnerability densities fall within a range, and for similar products they are closer together. We note that the ratio of vulnerabilities to the total number of defects is often in the range of 1% to 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 has presented plots 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 model given by Equation 5 is fitted to vulnerability data for the seven operating systems and the fit is found to be statistically significant. We also observe that 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

1. E. E. Schultz Jr., D. S. Brown and T. A. Longstaff, "Responding to Computer Security Incidents," Lawrence Livermore National Laboratory, ftp://ftp.cert.dfn.de/pub/docs/csir/ihg.ps.gz, July 23, 1990.
2. M. R. Lyu, editor., Handbook of Software Reliability Engineering, McGraw-Hill, 1995.
3. J. D. Musa, A. Ianino, K. Okumuto, Software Reliability Measurement Prediction Application, McGraw-Hill, 1987.
4. Y. K. Malaiya and J. Denton, "What Do the Software Reliability Growth Model Parameters Represent?" Proceedings IEEE International Symposium on Software Reliability Engineering, 1997, pp. 124-135.
5. Y. K. Malaiya and J. Denton, "Module Size Distribution and Defect Density," Proceedings IEEE International Symposium on Software Reliability Engineering, Oct. 2000, pp. 62-71.
6. P. Mohagheghi, R. Conradi, O.M. Killi and H. Schwarz, "An Empirical Study of Software Reuse vs. Defect-Density," Proceedings 26th International Conference on Software Engineering, 2004, May 2004, pp. 282-291.

7. A. Mockus, R.T. Fielding, and J. Herbsleb, "Two Case Studies of Open Source Software Development: Apache and Mozilla," ACM Transactions Software Engineering and Methodology, 11(3), 2002, pp. 309-346.
8. B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, D. Wright, "Towards Operational Measures of Computer Security," Journal of Computer Security, V. 2 (2/3), 1993, pp. 211-230.
9. S. Brocklehurst, B. Littlewood, T. Olovsson and E. Jonsson, "On Measurement of Operational Security," Proceedings of 9th Annual IEEE Conference on Computer Assurance, Gaithersburg, IEEE Computer Society, 1994, pp. 257-66.
10. W. A. Arbaugh, W. L. Fithen, J. McHugh, "Windows of Vulnerability: A Case Study Analysis," IEEE Computer, Vol. 33, No. 12, December 2000, pp. 52-59.
11. H. K. Browne, W. A. Arbaugh, J. McHugh, W.L. Fithen, "A Trend Analysis of Exploitation," Proceedings of IEEE Symposium on Security and Privacy, 2001, May 2001, pp. 214-229.
12. E. Jonsson, T. Olovsson, "A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior," IEEE Transactions on Software Engineering, April 1997, pp. 235-245.
13. B.B.Madan, K.Goseva-Popstojanova, K.Vaidyanathan, K.S.Trivedi, "Modeling and Quantification of Security Attributes of Software Systems," Proceedings of IEEE International Performance and Dependability Symposium (IPDS 2002), June 2002.
14. Eric Rescorla, "Is Finding Security Holes a Good Idea?", Proceedings Third Annual Workshop on Economics and Information Security (WEIS04), May 2004, pp. 1-18, http://www.dtc.umn.edu/weis2004/rescorla.pdf
15. Ross Anderson, "Security in Open versus Closed Systems – The Dance of Boltzmann, Coase and Moore," Conf. on Open Source Software: Economics, Law and Policy, Toulouse, France, June 2002, pp. 1-15, http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/toulouse.pdf
16. O. H. Alhazmi, Y. K. Malaiya, "Quantitative Vulnerability Assessment of Systems Software," Proceedings of International Symposium on Product Quality and Integrity (RAMS 2005), January 2005, pp. 14D3.1-6.
17. Ounce Labs, "Security by the Numbers: The Need for Metrics in Application Security," http://www.ouncelabs.com/library.asp, 2004.
18. ICAT Metabase, http://icat.nist.gov/icat.cfm, February 2004.
19. G. McGraw, "From the Ground Up: The DIMACS Software Security Workshop," IEEE Security and Privacy, March/April 2003. Volume 1, Number 2, pp. 59-66.
20. P. Rodrigues, "Windows XP Beta 02. Only 106,500 Bugs," http://www.lowendmac.com/tf/010401pf.html, Aug 2001.
21. O.S. Data, Windows 98, http://www.osdata.com/oses/win98.htm, March 2004.
22. The MITRE Corporation, http://www.mitre.org, February 2005.
23. Red Hat Bugzilla, https://bugzilla.redhat.com/bugzilla, January 2005.