Quantitative Vulnerability Assessment of Systems Software

Omar H. Alhazmi, Colorado State University
Yashwant K. Malaiya, Ph. D., Colorado State University

Key Words: security, operating systems, vulnerability, quantitative models.

SUMMARY & CONCLUSIONS

Operating systems represent complex interactive software systems that control access to information. Vulnerabilities present in such software represent significant security risks. In this paper, we examine the feasibility of quantitatively characterization of vulnerabilities. For Windows 98 and Windows NT 4.0, we present plots for cumulative numbers of vulnerabilities found. A time-based model for the total vulnerabilities discovered is proposed and is fitted to the data for two operating systems. We introduce a measure termed *equivalent effort* and propose an alternative model which is analogous to the software reliability growth models. We have shown that both models fit well and the fit is significant. We discuss the feasibility of using a new measure termed *vulnerability density*. We present the data on known defect densities for the two operating systems and discuss the relation between densities of vulnerabilities and the general defects. This relationship could lead us to potential ways of estimating the number of vulnerabilities in future.

1. INTRODUCTION

Connectivity of computing systems has brought security of operating systems under intense scrutiny. Much of the work on security has been qualitative, focused on detection and prevention of vulnerabilities in these systems. In this paper we address feasibility of quantitative characterization of security risks in terms of vulnerabilities present in the software. Such characterization of vulnerabilities can provide us metrics that can be used by the developers and potential users. It can be used to develop guidelines for allocation of resources for security testing, scheduling, and development of security patches. Furthermore, it can be used by the users for assessing risk and estimating needed redundancy in resources and procedures to handle potential breaches.

Quantitative characterization requires use of models that capture repeatable behavior. Such models have been in use in software reliability engineering field where the number of defects and the defect finding rate can be measured. In this paper we examine available data to identify possible approaches that may be applicable in practice.

Fortunately we have access to actual data about vulnerabilities found in major operating systems. In this paper we will try to address some major questions. Based on available data, do we observe any similarity of behavior when we observe new vulnerabilities discovery rate for different systems so that we can develop suitable models? It has been noted that for a given state of technology, the defect density of software systems at release tend to fall within a range (Ref. 1). This makes defect density a suitable measure for managing software projects. Is it possible to do something similar for vulnerabilities in the systems software? Here we identify metrics that are potentially measurable and can be used for managing risk in such systems.

Several quantitative methods have been proposed by the researchers to estimate the number of defects in a particular release version of software. These measures help in determining the resources that need to be allocated to test a particular piece of software. It would be very valuable to have similar methods specifically for addressing the potential vulnerabilities present. A vulnerability may be defined as a defect "which enables an attacker to bypass security measures" (Ref. 2). Since the operating system vulnerabilities – the faults associated with maintaining security requirements– are considered to be a special case of software defects, a similar measure for estimating security vulnerabilities is warranted. In this paper, we quantitatively examine the number of vulnerabilities in two operating systems using the databases that track the vulnerabilities reported.

Researchers in software reliability engineering have developed methods that use plots of cumulative number of defects found against time. Methods have been developed to project the mean time to failure (MTTF) that will result after a specific testing time (Refs. 2-4). Very little quantitative work has been done to characterize security vulnerabilities along the same lines. One major difference makes understanding the vulnerability discovery rate harder. Throughout the lifetime of the software since its release, it encounters changes in its usage environment. When the new version of an operating system is released, its installed base starts to grow. Meanwhile, the number of installations of the older version starts to decline. The rate at which the vulnerabilities are discovered are influenced by the interaction between the system and the environment. We need to look at both to form a comprehensive perspective of security.

Quantitative evaluation of security has different scopes. Depending on how we view systems, the applicable metric can be different. In (Refs. 5, 6), Littlewood et al. have proposed measures to assess security from a reliability prospective. They have compared the reliability and security attributes of systems. They have proposed the usage of effort instead of time to characterize the accumulation of vulnerabilities. However, they did not specify how to assess effort. A standard terminology in this field has not yet emerged. In (Ref. 7) the authors proposed an operational security evaluation in UNIX systems using Markov modeling. Researchers (Ref. 8) have identified some common vulnerability types and discussed how designers can avoid them. Arbach et al. (Ref. 9) and Browne et al. (Ref. 10) have examined several systems using the incidents and vulnerability data reported by Computer Emergency Readiness Team (CERT). Browne et al. examined some actual statistical data to study the trends in incidents occurring. They have suggested that the cumulative number of incidents is linearly related to square-root of time. Shim et al. (Ref. 11) have applied a software reliability growth model to the incidents data. Alhazmi et al. (Ref. 12) have presented plots for cumulative vulnerabilities for some Windows and UNIX based operating systems.

In section 2 we present available data related to some of the operating systems as well as their usage environment. We introduce a time-based model in sections 3. An effort-based model is presented in section 4. Then, in section 5, we fit the models and we analyze the goodness of fit. The concept of *Vulnerability density* is introduced and its relation to defect density is examined using available data in section 6.

2. *VULNERABILITIES IN TODAY'S ENVIRONMENT*

Operating Systems face escalating security challenges because connectivity is growing and the overall number of incidents is increasing. CERT and other databases keep track of reported vulnerabilities. An increasing number of individuals and organizations depend on the internet for financial and other critical services. That has made potential exploitation of vulnerabilities very attractive to criminals with suitable technical expertise.

The Microsoft's Windows family of operating systems has dominated the Operating System (OS) market for the past decade. Records of each discovered vulnerability is available. Therefore, they are good examples of complex systems to study.

In this paper we examine the data (Ref. 13) for Windows 98 and Windows NT 4.0. The accumulated numbers of vulnerabilities for both of them are shown in Figure 1. We should note that Windows NT 4.0 was designed as a server operating system, while Windows 98 was intended to be a client operating system. Figure 1 shows a common pattern. They begin with a slow start and then rise with steeper slope. Eventually they show saturation. This behavior serves as the basis for the model presented in the next section.

3. *A TIME-BASED KNOWN-VULNERABILITY DISCOVERY MODEL*

Each operating system passes through several phases: the release of the system, increasing popularity, peak and stability followed by decreasing in popularity that ends with the system eventually becoming obsolete. From Figure 1 we can see a common pattern of three phases in the cumulative vulnerabilities plot for a specific version of an OS, as identified in Figure 2.

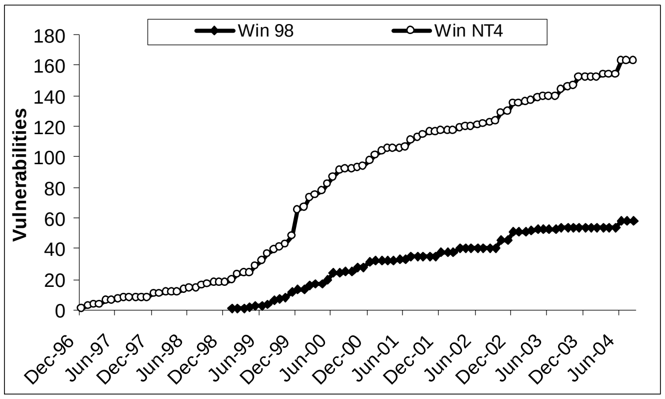

Figure 1 - The Cumulative Number of Vulnerabilities of Windows 98 and Windows NT 4.0

During these three phases the usage environment changes and that impacts the vulnerability detection effort. There is a **learning phase**, in which the software testers (including hackers and crackers) begin to understand the target system and gather the knowledge needed about the system to break it successfully. After the learning phase, the new system will start attracting a significant number of users and begin to face stronger challenges. This continues to increase until the operating systems reaches the peak of its popularity. This is termed as the **linear phase** since the number of vulnerabilities tends to grow linearly. It will stay there for some time until the system starts getting replaced by newer systems. Then, the technical support for that version and hence the frequency of update patches will begin to decline. The users start to switch or upgrade to a more modern system. At the same time, attackers start to lose interest in the system, because they are usually attracted to the most recent technology. This is termed the **saturation phase**.

The learning phase may be very brief if the adaptation of the OS is fast. The linear phase is the most important phase since this is when most of the vulnerabilities will be found. Saturation may not be seen if a significant number of vulnerabilities are still present and continue to be found.

Here we propose a model describing the relationship between cumulative vulnerabilities and the calendar time. This is somewhat similar to the reliability growth models; however we recognize that there is significant change in the effort spent in finding the vulnerabilities. This effort is small in the learning phase and grows as the OS becomes more common. This effort drops again in the saturation phase.

In this model we assume that the rate of change cumulative number of vulnerabilities *y* is governed by two factors, as given in Equation 1 below. One of the factors declines as the number of undetected vulnerabilities remaining decline. The other factor rises with time to take into account the rising share of the installed base. The saturation effect is modeled by the first factor. It is possible to obtain a more complex model, but this model provides a good fit to the data as shown below.

Let us assume that the vulnerability discovery rate is given by the differential equation.

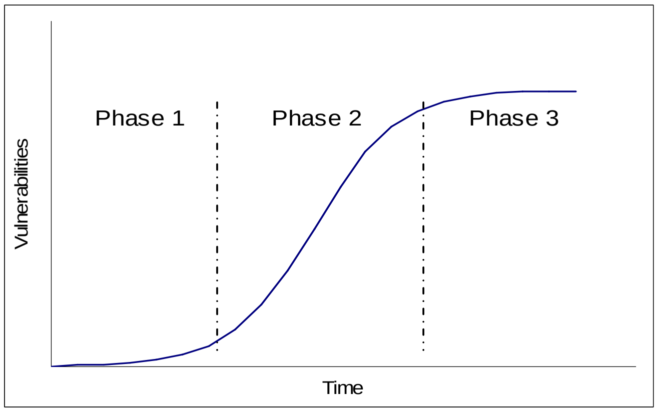

Figure 2 - The Basic 3-phase S-shaped Model

$$\frac{dy}{dt} = Ay\,(B - y) \qquad (1)$$

Where y is the cumulative number of vulnerabilities, t is the calendar time, initially t=0. A and B are empirical constants determined from the recorded data. By solving the differential equation we obtain

$$y = \frac{B}{BCe^{-ABt} + 1} \qquad (2)$$

While C is a constant introduced during solving Equation 1. It is thus a three-parameter model. In Equation 2, as t approaches infinity, y approaches B. Thus, the parameter B represents the total number of accumulated vulnerabilities that would eventually be found. The model given by Equation 2 will be referred to as the **time-based model**.

This model assumes that the vulnerabilities found in an operating system depend on its own usage environment. It should be noted that phase 3 may not be seen in an OS which has not been around for a sufficiently long time. Also in some cases phase 1 may not be significant if the initial adaptation is quick due to better prior publicity.

4. *AN EFFORT-BASED KNOWN-VULNERABILITY DISCOVERY MODEL*

Vulnerabilities are reported using calendar time intervals. The reason behind this is that it is easy to record vulnerabilities and link them to the time of discovery. This however does not consider the changes occurring in the environment during the lifetime of the system. A major environmental factor is the number of installations which depends on the share of the installed base of the specific system. It is much more rewarding to exploit vulnerabilities that exist in a large number of computers.

It can be expected that a larger share of the effort going into discovery of vulnerabilities, both in-house and external, would go toward a system with larger installed base.

Using effort as a factor was first discussed in (Refs. 5-6). However, they have not suggested a unit or a way to measure effort. Here, we examine the use of a measure termed *Equivalent Effort* (*E*) which is calculated using

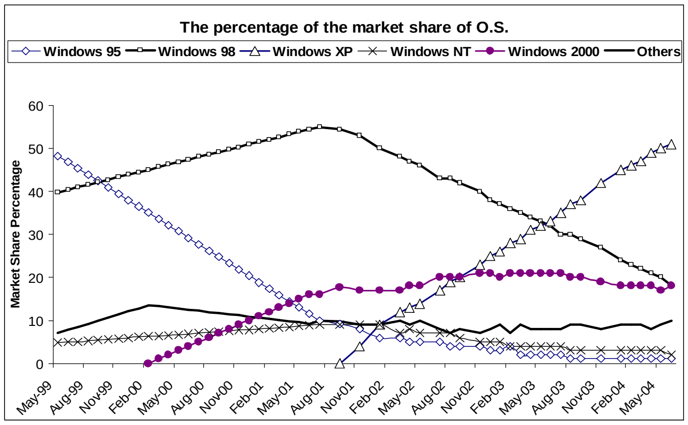

Figure 3 - The Percentages of Installed base of some of the Windows Systems.

$$E = \sum_{i=0}^{n} (U_i \times P_i) \qquad (3)$$

Where U_i is the total number of users of all systems at the period of time *i*, and P_i is the percentage of the users using the system that we are measuring its E (see Figure 3). Here we use a month as a unit of time and thus E would be in *users-months*. Using available data (Ref. 12), E can be calculated for the windows operating systems (Ref. 12). Now let us use another model that uses effort as a factor to model vulnerability discovery. Equivalent effort reflects the effort that would have gone into finding vulnerabilities more accurately than time alone. This is somewhat analogous to using CPU time for Software Reliability Growth Models (SRGMs) (Ref. 3). If we assume that the vulnerability detection rate with respect to effort is proportional to the fraction of remaining vulnerability, then we get an exponential model, just like the exponential SRGM. The model can be given as follows:

$$y = B\,(1 - e^{-\lambda_{vu} E}) \qquad (4)$$

Were λ_{vu} is a parameter analogous to failure intensity in SRGMs and *B* is another parameter. *B* represents the number of vulnerabilities that would eventually be found. We will refer to the model given by Equation 4 as the **Effort based model**.

5. *APPLICATION OF THE TIME AND EFFORT BASED MODELS*

In this section we apply both the time-based and the effort-based models proposed in the pervious section to the data for Windows 98 and Windows NT 4.0. We used vulnerability data for Windows 98 from May 1999 and to October 2002, and for NT 4.0 from August 1996 to April 2003.

Figure 4 shows the Windows 98 data fitted to the time-based model given by Equation 2. The data was fit using least squares method for estimating the three parameters. To examine applicability of the effort-based model given by Equations 3 and 4 to Windows 98, we used the same vulnerability data for Windows 98 along with the corresponding data on the share of installed base and the numbers of internet users. Figure 6 gives the data along with the fitted model. The corresponding plots for NT 4.0 are given in Figures 5 and 7. Table 1 gives the parameter values obtained. For chi square goodness of fit test, we chose alpha level as 5%. The chi square values are given in Table 1. The chi square values are less than the critical values and therefore the fit is significant.

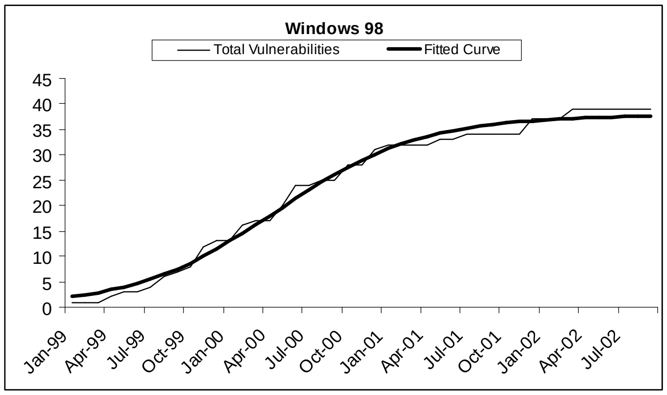

Figure 4 - Fitting Windows 98 Data (Time-Based Model)

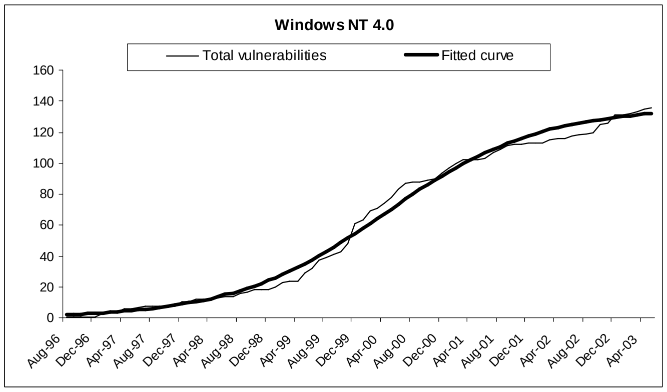

Figure 5 - Fitting Windows NT 4.0 Data (Time-Based Model)

For the two operating systems, both models fit well. The time-based model is easier to use because it does not require evaluation of E. However the effort based model removes the variability due to change of effort during OS life-time.

For both OSs, from plots in Figures 4-7, we observe a departure from the model near the end of the period. This is due to the fact that the next version of the OS, had gained a significant share of the installed base, and it shared code with the previous version. Some of the vulnerabilities reported for Windows 98, were actually found in Windows XP.

Further research is needed to develop composite models to take this into account. We have applied both models to the data for other Windows operating systems. In all cases, acceptable fit is obtained. However in some cases, there is significant overlap among successive versions (Ref. 12), and composite models are likely provide better accuracy.

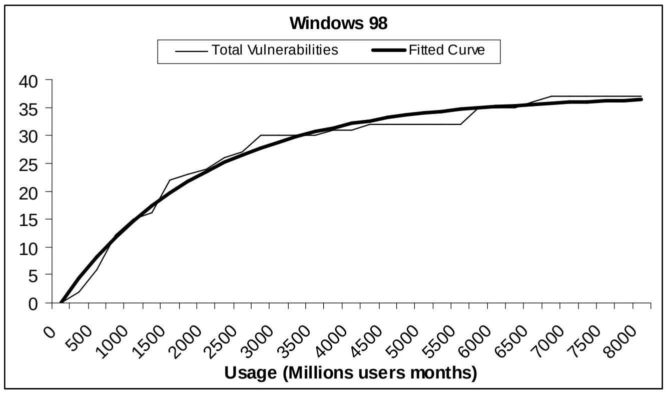

Figure 6 - Fitting Windows 98 Data (Effort-Based Model)

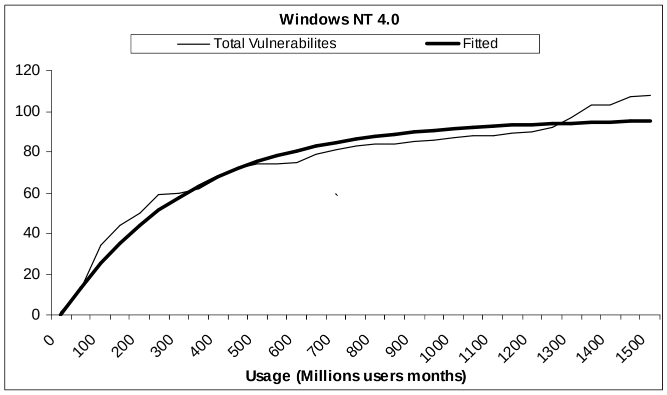

Figure 7 - Fitting Windows NT 4.0 Data (Effort-Based Model)

Time Based Model		
	Windows 98	**Windows NT 4.0**
A	0.004873	0.000692
B	37.7328	136
C	0.5543	0.52288
χ^2	7.365	35.584
$\chi^2_{critial}$	60.481	103.01
P-value	$1- 7.6\text{x}10^{-11}$	0.9999973
Effort Based Model		
B	37	108
λ_{vu}	0.000505	0.003061
$\chi 2$	3.510	15.05

χ2critia l	44.9853	42.5569
P-value	1- 3.3x10-11	0.985

Table 1 - Chi-square Analysis of Goodness of Fit

6. *MEASURING SYSTEMS VULNERABILITY DENSITY*

We now introduce the notion of vulnerability density, a metric that can be calculated when we have the needed statistics about the system. From these statistics we can compute this metric that enables us to compare the systems and assess their maturity with respect to security vulnerabilities.

This metric is primarily concerned with the number of vulnerabilities and its relation to the size of the system. To measure a system's size we have two options. We can use the size of the installed system in bytes; the advantage of this measure is that information is readily available. However, the measure may vary from one installation to another. The other measure is number of source lines of code. We choose this measure for its simplicity and popularity in the software engineering domain.

Definition: Vulnerability density is the number of vulnerabilities per unit size of code.

The significance of studying *vulnerability density* rather than the absolute number of vulnerabilities is to get a real assessment of the whole picture. A conceptually similar notion is defect density (Ref. 14), a measurement that has been used for several years and that has become a common measurement in software reliability and dependability field. Vulnerability density allows us to normalize the number of vulnerabilities by dividing them by the size of the system. Density is better measure than the absolute number. When two systems, one large and one small, have the same density, they can be regarded to have similar maturity with respect to the dependability. If the instruction execution rates and other system attributes are the same, a system with a higher defect or vulnerability density is likely to fail more often.

Measuring the exact vulnerability density would require us to know the number of all vulnerabilities of the system. Because this is not the case, we will calculate *known vulnerability density*, which is based on the reported vulnerabilities. In future we may be able to estimate the number of undiscovered vulnerabilities using historical data; this would require further development of the quantitative vulnerability models and collection of sufficient data. Known vulnerability density V_{KD} can be defined as the reported number of vulnerabilities in the system per unit size of the system. This is given by:

$$V_{KD} = \frac{V_K}{S} \qquad (5)$$

Where S is the size of software and V_K is the reported number of vulnerabilities in the system. Table 2 presents the values based on data reported in (Ref. 12) based on data from several sources. It gives the known defect density D_{KD}, V_{KD} and the ratios of the two.

In Table 2 we see that the source code size is respectively 16 and 18 MSLOC (Million source lines of code) for NT and Win 98, approximately the same (Ref. 15). The reported defect densities at release, 0.625 and 0.556, are also similar for the two OSs. The known vulnerabilities in Table 2 are as of July 2004 (Ref. 13). We notice that for Windows 98, the vulnerability density is 0.0032 whereas for Windows NT 4.0 it is 0.0101, significantly higher. The higher values for NT 4.0 may be due to two factors. First, as a server OS, it may contain more code that handles access mechanisms. Secondly because attacking servers would generally be much more rewarding, it must have attracted a lot more testing effort resulting in detection of more vulnerabilities.

The last column in Table 2 gives the ratios of known vulnerabilities to known defects. For the two OSs the values are 1.62% and 0.58%. Longstaff (Ref. 16) has hypothetically assumed that vulnerabilities can be 5% of the total defects. On the other hand, Anderson (Ref. 17) assumes the ratio to be about 1%, although neither of the two used actual data. The values given in Table 2 suggest that Anderson's assumption may be closer to reality. However we need to evaluate similar ratios for other OSs before we can conclude what typical ranges of ratios should be expected.

System	MSLOC	Known Defects (1000s)	D_{KD} (/Kloc)	Known Vulner - abilies	V_{KD} (/Kloc)	Ratio V_{KD} /D_{KD}
NT 4.0	16	10	0.625	162	0.0101	1.62%
Win 98	18	10	0.556	58	0.0032	0.58%

Table 2 - Known Defect Density vs. Known Vulnerability Density.

Table 2 presents numbers for only two operating systems. When we have enough data, such numbers can serve as useful references. Just as we have in the past asked the question "when to stop testing?" for defects, we need to ask when are we reasonably sure that a system software is reasonably free of vulnerabilities. This would require development of accurate and validated models along with suitable threshold values. As we know more about potential vulnerabilities, the typical vulnerability densities should decline, just like typical defect densities have declined over the years. Further research is needed to establish threshold vulnerability densities and to assess the risk level associated with a specific value.

REFERENCES

1. Y. K. Malaiya and J. Denton, "What Do the Software Reliability Growth Model Parameters Represent?", *Int. Symp. on Software Reliability Engineering*, 1997. pp. 124-135.
2. E. E. Schultz Jr., D. S. Brown and T. A. Longstaff, Responding to Computer Security Incidents, Lawrence Livermore National Laboratory, July 23, 1990.

3. M. R. Lyu Ed., *Handbook of Software Reliability Engineering*, McGraw-Hill 1995.
4. J. D. Musa, A. Ianino, K. Okumuto, *Software Reliability Measurement Prediction Application*, McGraw-Hill, 1987.
5. S. Brocklehurst, B. Littlewood, T. Olovsson and E. Jonsson, "On Measurement of Operational Security". *Proc. 9th Annual IEEE Conference on Computer Assurance*, Gaithersburg, pp. 257-66, IEEE Computer Society, 1994.
6. 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), pp 211-230, 1993.
7. M. Dacier, Y. Deswarte and M. Kaâniche, Quantitative Assessment of Operational Security: Models and Tools, Technical Report, LAAS Report 96493, May 1996.
8. I. Krusl, E. Spafford, M. Tripunitara, Computer Vulnerability Analysis, Department of Computer Sciences, Purdue University. COAST TR 98-07 1998.
9. W. A. Arbaugh, W. L. Fithen, J. McHugh, "Windows of Vulnerability: A Case Study Analysis**",** *IEEE Computer*, Vol. 33, No. 12, pp. 52-59, December 2000.
10. H. K. Browne, W. A. Arbaugh, J. McHugh, W.L. Fithen, "A Trend Analysis of Exploitation", *Proc. IEEE Symposium on Security and Privacy*, 2001, May 2001, pp. 214 – 229.
11. C.Y. Shim, R.E. Gantenbein, and S.Y. Shin, "Developing a Probabilistic Security Measure Using a Software Reliability Model", Information, October 2001, pp. 409-418.
12. O. H. Alhazmi, Y. K. Malaiya, I. Ray, Vulnerabilities in Major Operating Systems, Technical Report, Computer Science Department, Colorado State University, 2004.
13. ICAT Metabase, http://icat.nist.gov/icat.cfm, September 12, 2004.
14. Y. K. Malaiya, P. K. Srimani, "Software Reliability Models: Theoretical Development, Evaluation and Applications", *IEEE Computer Society Press*, 1990.
15. G. McGraw. "From the Ground Up: The DIMACS Software Security Workshop", *IEEE Security & Privacy*. March/April 2003. Volume 1, Number 2. pp. 59-66.
16. T. Longstaff, "CERT Experience with security problems in Software", Carnegie Mellon Univ., http://research.microsoft.com/projects/SWSecInstitute/slides/Longstaff.pdf, June 2003.
17. R. Anderson; "Why information security is hard – An Economical Prospective", *Proceedings of the 17th annual Computer Security applications conference*, 2001, P358.

BIOGRAPHIES

Omar H. Alhazmi
Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873 USA.

e-mail: omar@cs.colostate.edu

Omar Alhazmi is currently a Ph.D. student at Colorado State University since January 2002. He received his Master's degree in computer science from Villanova University in 2001.

Yashwant K. Malaiya, *Ph.D*
Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873 USA.

e-mail: malaiya@cs.colostate.edu

Yashwant K. Malaiya is a Professor in Computer Science Department at Colorado State University. He received MS in Physics from Sagar University, MScTech in Electronics from BITS Pilani and PhD 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 2003 IEEE International Symposium on Software Reliability Engineering. He is a senior member of IEEE.