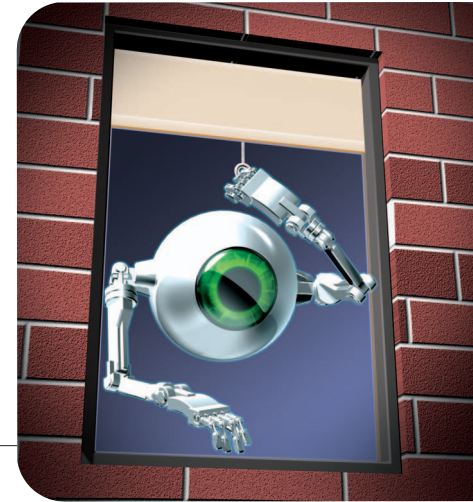


Estimating Software Vulnerabilities

Any given piece of software has some number of publicly disclosed vulnerabilities at any moment, leaving the system exposed to potential attack. A method for identifying and analyzing these vulnerabilities uses public data from easily accessible sources.



JEFFREY R. JONES
Microsoft

As long as products have vied for selection by customers, analysts have calculated metrics to compare them. Which car has the most horsepower or achieves the top speed? Or, as times changed, which car gets the best gas mileage? In software, many criteria have achieved a level of importance, and as software security issues have grown, the industry has worked to develop comparison metrics. Much of this work uses measurements that are based on issues a vendor has addressed or fixed. Vendors post security notifications, for example, to notify users that a patch is available to address a known vulnerability. You can calculate days of risk by subtracting the date an issue was publicly disclosed from the date the vendor made a patch available.

But what about the security issues that the vendor hasn't fixed? In other words, on any given day, how many security flaws are publicly disclosed and potentially available for malicious use by hackers? The amount of work necessary to create and maintain a database for all public disclosures affecting components of complex modern systems would be substantial. Although some sites, such as Secunia (www.secunia.com), report a specific number of unpatched issues for software products, can you be confident that the information is comprehensive and accurate?

I've developed a method for identifying and charting software exposure to unpatched vulnerabilities. I then analyzed the vulnerability data to determine accuracy. Users or vendors can use this methodology to interpret exposure data and apply it practically.

Software vulnerabilities' role in performance metrics

Jim Reavis was the first to use vendor response times to

compare security response times for vendors.¹ He compared Sun, Red Hat, and Microsoft security response times using data gathered from vendors' security advisories, as well as data from well-known security consultants, CERT, hacker groups, and mailing lists such as Bugtraq. Later that year, William Arbaugh more formally defined a vulnerability life cycle's stages, laying the groundwork for subsequent comparison studies.²

The Forrester Group performed an analysis similar to Reavis's, comparing Microsoft and four Linux distribution vendors, again using data from similar sources over a one-year period.³ Whereas the Forrester study looked at all issues by vendor, later studies by Richard Ford and his colleagues⁴ and Herbert Thompson and Fabien Castellan⁵ examined the available vulnerability data as it applied to specific products installed and configured as a Web server and a database server, respectively. Both studies sought to measure:

- the number of security advisories vendors issue,
- the number of software vulnerabilities vendors fixed, and
- the average time a vendor takes to provide a patch for a publicly disclosed vulnerability.

Brian Witten and his colleagues asked whether open source improves system security, referencing the Reavis data. They concluded that the data shows the open source period "experienced less exposure in the form of known but unpatched vulnerabilities over a 12-month period than was experience by either of two proprietary counterparts."⁶

Although the earlier studies use software vulnerabilities as the basis for their measurements, they aren't explicit about which vulnerabilities they studied. Ford and his colleagues, on the other hand, specifically state which vulnerabilities they include in their study of Web server roles⁴:

“For our Web server role comparison, we will use vulnerabilities disclosed earlier than 2004 if and only if the solution vendor (Microsoft or Red Hat) has released a fix for these issues in 2004. Similarly, we will not consider vulnerabilities announced in 2004 but fixed in 2005.”

It seems likely that the study would have analyzed publicly disclosed but unpatched vulnerabilities during 2004 if the researchers could have generated a definitive list.

Some public sources—notably Secunia—do provide information about unpatched security issues. For example, on 31 March 2005, the Secunia site showed that for the Red Hat Enterprise Linux 3 Advanced Server, zero out of 152 Secunia advisories were marked as unpatched in the Secunia database. In addition, for the Windows Server 2003 Enterprise Edition, six out of 45 Secunia advisories were marked as unpatched in the Secunia database.

Does the number of unpatched Secunia advisories reflect, or is it proportionate to, the set of publicly disclosed vulnerabilities available to malicious attackers? It's almost immediately apparent that it can't be.

Secunia advisories largely reflect vendor advisories. So, one Secunia advisory reflects Microsoft's advisory MS04-011, in which the company addressed 14 discrete vulnerabilities. Similarly, one Secunia advisory reflects Novell's SUSE-SR:2005:003, which addresses more than 45 vulnerabilities. Did Secunia show 45 unpatched issues on the day before Novell released SUSE-SR:2005:003? It did not, even though many other Linux distributions had already addressed and publicly discussed those issues. No one currently provides that level of attribution to the complex Linux-based operating system distributions.

The power of 20/20 hindsight

A disclosed vulnerability can be in one of two states:

- publicly known, with no patch available from the vendor, or
- publicly known, with a patch available from the vendor.

Metrics from previous studies are driven by the latter; however, the former might be much more interesting if we can establish an accurate way to compare them.

With one known exception (in March 2003, Microsoft provided a patch for Windows XP and Windows 2000, but announced that it couldn't provide a patch for Windows NT 4.0 without rearchitecting the operating system to the point of making the fix's use impossible for

the legacy NT customers that needed it), vendors will fix nearly all vulnerabilities over time for actively supported products, either with a patch or via a required product upgrade to a minor release or service pack. (Once a product life cycle ends, however, it's unlikely that vendors will backport issues found in successor products to unsupported platforms. Many vendor state this explicitly.) With this assumption in mind, if we go far enough back in time, we can use the past announcement of fixes to determine what public issues were unfixed on any given day.

I constructed a table of all vulnerabilities affecting Windows 2000 Server that were fixed during the year 2001. I also included any vulnerability that was fixed after 2001, but for which the vulnerability was publicly known prior to the end of 2001. I listed the vulnerabilities by Mitre vulnerability ID (Mitre maintains a list of vulnerabilities and exposures at <http://cve.mitre.org>. The common vulnerabilities and exposures [CVE] ID or CVE name is a unique 11-digit encoding. The first three digits are CAN [for candidate] when first assigned and change to CVE when confirmed. The next four digits are the year, and the last four digits represent the number NNNN for the Nth vulnerability candidate for the year.)

To construct the table, I followed Ford and colleagues⁷ step-by-step methodology. First, I built a table of all vulnerabilities fixed in Windows 2000 Server through the end of 2005. Next, I filtered out any vulnerability that was disclosed publicly after 31 December 2001 or fixed prior to 1 January 2001. The table contains 60 unique vulnerabilities, of which 58 were fixed during calendar year 2001 and two were fixed sometime later. To date, five years later, no other vulnerability that was disclosed prior to the end of 2001 has been fixed.

Using this table as a data source, I use a simple conditional summation formula to calculate how many vulnerabilities were public but had no corresponding vendor patch on each day of the year. I refer to this value as the *daily vulnerability exposure* (DVE), and calculate it as:

$$DVE_{date} = \sum_{vulns} (firstpublic < date) \text{ and } (datefixed > date)$$

The formula expresses that, on any given day, the software would be exposed to a vulnerability if the vulnerability were publicly disclosed prior to that day, but a vendor fix wasn't available until after that day.

Next, I calculate the DVE for each day of the year and graph the DVEs as an exposure chart, as Figure 1 shows.

As the chart shows, the DVE for Windows 2000 Server reached a peak of six unpatched vulnerabilities during January 2001. Microsoft drove the DVE down to zero in November, having fixed all known vulnerabilities before more were publicly disclosed in December. The graph also shows that two disclosed vulnerabilities were un-

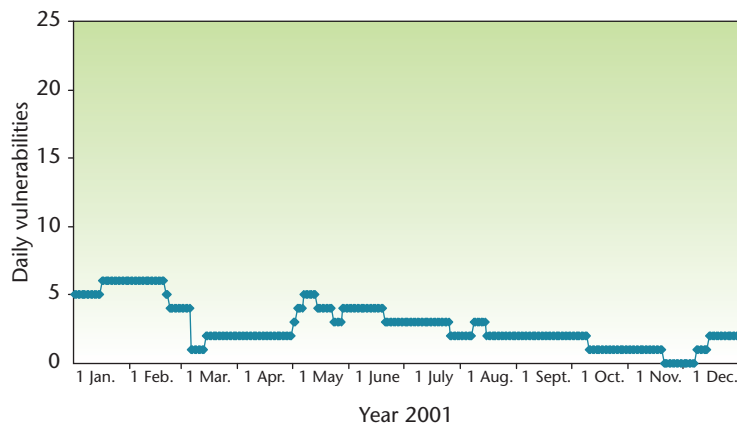


Figure 1. Windows 2000 exposure chart for 2001. The chart shows the daily vulnerability exposure (DVE)—that is, vulnerabilities disclosed by 31 December 2001 and fixed as of 1 January 2006—for each day of 2001.

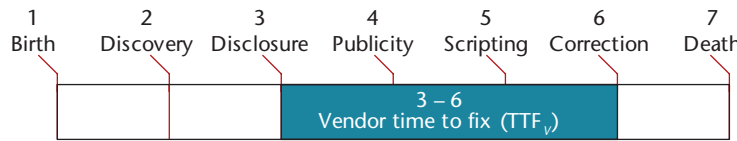


Figure 2. Vulnerability life-cycle diagram. I refer to the period from disclosure to correction as “vendor time to fix” or TTF_v .

patched as the year ended (from hindsight, we know these were later fixed in February and April 2002).

New issues are being discovered and disclosed all the time, and vendors periodically release patches to close issues. How does the exposure chart model handle all this activity? Figure 1 shows, for example, that two vulnerabilities were disclosed and unpatched during all of April 2001. This could represent the same two vulnerabilities for the entire month, but it could also describe a scenario in which the vendor fixed two issues each day and two new issues were disclosed each day. You can’t tell from the chart. Moreover, from a system exposure perspective, it doesn’t matter much because two unpatched issues are available for malicious attackers to exploit on any given day.

Vulnerability timeline and accuracy

Using data from five years ago gives me confidence that my estimate for DVE on any given day is accurate. Next, we must consider how recently we can apply this method and still have confidence in the estimate. Two years is probably still a reasonable estimate. One year? Six months? Three months? One month? I don’t know, but let’s see if we can develop a timeframe for confidence.

As noted earlier, on the last day of 2001, the vulnera-

bility chart shows two publicly disclosed but unpatched vulnerabilities. If I apply my method at any time after 4 April 2002, I’d generate the same chart that I get looking back with perfect hindsight from 2005. The vulnerability that was fixed on 4 April (CAN-2002-0051) was first publicly disclosed on 7 December 2001 (see <http://cert.uni-stuttgart.de/archive/bugtraq/2001/12/msg00080.html>), and was public for 118 days before Microsoft made a patch available to users. So, should we always wait 118 days? Was CAN-2002-0051 the last vulnerability made public in 2001? It wasn’t. In fact, Microsoft disclosed CAN-2002-0057 after CAN-2002-0051 and fixed them in less time. Would examining each disclosed vulnerability help us determine a timeframe within which we could be confident of the DVE estimation?

Vendor time to fix

Arbaugh and his colleagues defined a vulnerability life-cycle model that has become a common reference for vulnerability discussions.² They define seven stages: *birth*, *discovery*, *disclosure*, *publicity*, *scripting*, *correction*, and *death* (see Figure 2). Laura Koetzle and her colleagues conducted a year-long comparison focused on the period of increased security risk between disclosure and correction, and called this time period “days of risk.”³ For my purposes, I use the term “vendor time to fix,” or TTF_v , for the disclosed vulnerability.

Examining the historical performance of a vendor’s TTF_v statistically should help us determine how long to wait after a given date to establish reasonable accuracy.

Using the 60 vulnerabilities disclosed before the end of 2001 and fixed during that year, I generated a TTF_v histogram, as Figure 3a shows. Additionally, I’ve charted the cumulative distribution function (CDF) across the histogram, representing the probability that a vulnerability will have a TTF_v less than that many days. The CDF shows that based on the Microsoft TTF_v values for 2001, 91 percent of vulnerabilities will be fixed after 108 days, and 96 percent of vulnerabilities will be fixed after 162 days. This histogram doesn’t include the two vulnerabilities we know will be fixed later in 2002, but it does represent a statistical possibility that a few more have already been disclosed and will be fixed during the next 162 days.

Using this information, we would have confidence that on the 163rd day of 2002, or 13 June 2002, an exposure chart for 2001 would include more than 96 percent of vulnerabilities.

Next, let’s take a more recent data set and see if we can apply the same methodology and then test it by looking to see what happened.

I compiled the set of vulnerabilities fixed on Windows 2000 Server during 2004, which gave me 60 unique vulnerabilities. Charting the TTF_v histogram (see Figure 3b), we get a similar outcome to 2001—that is, the vendor is likely to have fixed 96.6 percent of vulnerabilities in 142 days.

Applying this information backward, on the last day of 2004, we have confidence that a DVE chart would represent 96 percent of vulnerabilities through 10 August 2004. Applying the information forward, we should be able to have a DVE chart containing 96 percent of vulnerabilities by 22 May 2005.

We can check this information again by jumping forward to 22 May 2005 and comparing it to the full set of information we have today. By 22 May 2005, Microsoft had fixed an additional 10 vulnerabilities that had been disclosed prior to the end of 2004. Microsoft fixed at least one additional vulnerability in June that was disclosed prior to the end of 2004. This means that by 22 May, the chart accounted for 70 out of 71, or 98.6 percent of vulnerabilities known through 1 January 2006.

Figure 4 shows the updated DVE chart and the exposure chart as it existed at the end of 2004 and as of July 2005, after Microsoft had fixed the last known vulnerability disclosed prior to the end of 2004.

Potential uses of an exposure chart

Up to this point, I've established a repeatable method for generating an exposure chart and substantiated a timeframe for reasonable accuracy, and posited that users or administrators might use this type of chart to compare different software's security vulnerability factors. In this manner, an exposure chart shows whether a product is vulnerable and how many issues contribute to that exposure.

An exposure chart can also be a useful tool for software vendors, who can use it to examine how their security response process responds to the community of security researchers who find and publicly disclose software vulnerabilities. An understanding of this process could help vendors in resource planning. An exposure chart that trends up indicates that a vendor response process isn't keeping up with the rate of vulnerabilities being disclosed. The vendor could apply more resources to drive the trend back down, which would indicate that they were fixing issues as fast as they were being disclosed.

We could apply further filters using severity. We could take a severity rating, such as the vendor severity rating or the National Vulnerability Database's (<http://nvd.nist.gov/download.cfm>) objective rating to examine exposure of the more severe security vulnerabilities. The National Vulnerability Database recently assigned Common Vulnerability Scoring System (CVSS) scores to each vulnerability in its database, offering an objective way to filter for severity.

Methodology limitations

One weakness in using this methodology for estimating DVE and charting exposure is that it depends on vendors actively maintaining their products and assumes that vendors responsibly fix disclosed issues in their software, at

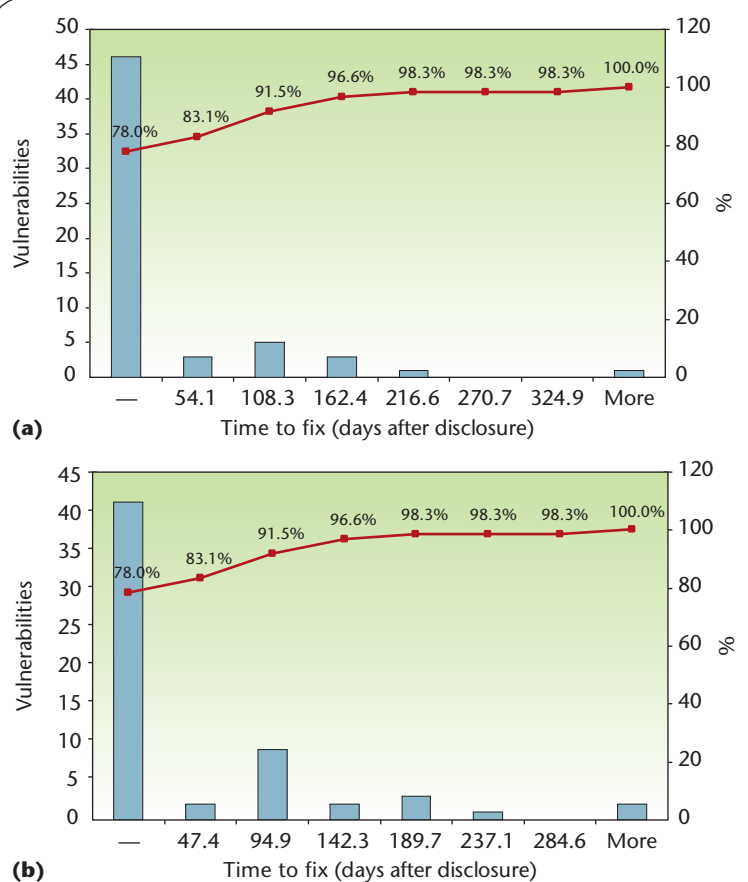


Figure 3. Windows 2000 Server time-to-fix (TTF) histogram for all vulnerabilities disclosed and fixed by (a) 31 December 2001 and (b) 31 December 2004 (as known on 1 January 2006). In both cases, the exposure chart shows that the vendor is likely to have fixed roughly 96 percent of vulnerabilities within 163 days.

least over time. In other words, issues that are publicly disclosed but that the vendor never fixes won't show up in an exposure chart.

However, this limitation might have less practical impact than it does in theory because customer pressure will force most vendors to fix publicly disclosed issues over time. Additionally, many vendors have agreed to follow the recommendations in the National Infrastructure Advisory Council's Vulnerability Disclosure Framework (www.dhs.gov/interweb/assetlibrary/vdwdgreport.pdf) on managing the vulnerability lifecycle, which results in disclosure as patches are made available. Even assuming that a vendor might intentionally or unintentionally omit some vulnerabilities for a given product, a DVE chart using my methodology would still establish a solid minimum exposure that would be useful for validating other sources of information about unpatched vulnerabilities.

Vendor response and TTF performance can lag or improve over time. In the longer term, the exposure charts

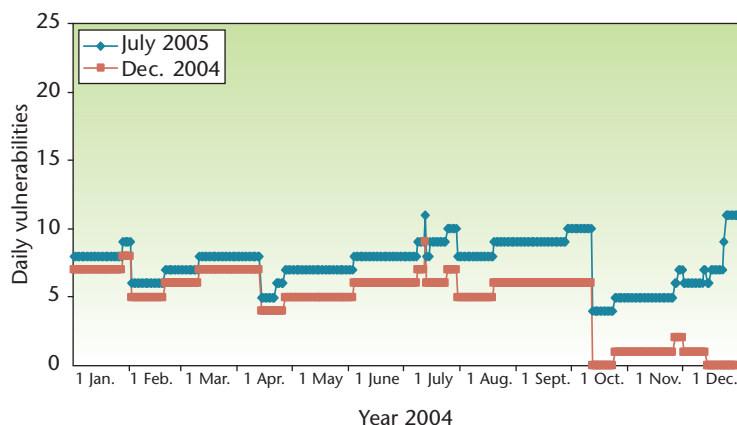


Figure 4. Windows 2000 exposure chart for 2004 (using data through January 2006). The chart compares the number of vulnerabilities disclosed prior to the end of 2004 that Microsoft fixed by the end of that year and as of July 2005.

will demonstrate these changes with an increase or decrease in the exposure trend. In the shorter term, we might want to drive the statistical analysis from a recent subset of data to accommodate response performance changes. Note that my previous examples used one-year periods to drive the statistical analysis rather than the entire history of data available for the product. In cases where a vendor has a large average TTF, a year might not be enough. This is an area for further research.

From here, I hope to leverage the method as a foundation for exploring DVE trends over time and providing quantitative comparisons of different software types. I'd also like to explore differences in similar software offerings' exposure to examine the question Reavis posed in 2000—that is, which vendor fixes security problems faster—and see what exposure charts for modern x86 operating systems—such as Windows Server 2003, FreeBSD, or Red Hat Enterprise Linux 4—might show. I doubt it will end the Windows versus Linux security debate, but perhaps it will stir some new discussions. □

References

1. J. Reavis, "Linux vs. Microsoft: Who Solves Security Problems Faster?" *CSOInformer*, 17 Jan. 2000, <http://csoinformer.com/research/solve.shtml>.
2. W. Arbaugh, W. Fithen, and J. McHugh, "Windows of Vulnerability: A Case Study Analysis," *Computer*, vol. 33, no. 12, 2000, pp. 52–59.
3. L. Koetzle et al., "Is Linux More Secure than Windows?" Forrester Research, 2004; www.forrester.com/Research/Document/Excerpt/0,7211,33941,00.html.
4. R. Ford, H. Thompson, and F. Casteron, "Role Comparison Report: Web Server Role," Mar. 2005; www.securityinnovation.com/pdf/windows_linux_final_study.pdf.
5. H. Thompson and F. Casteron, "Get the Facts: Role Comparison Report: Database Server Role," Mar. 2005; www.microsoft.com/windowsserversystem/facts/analyses/sirolecomparison.msp.
6. B. Witten, C. Landwehr, and M. Caloyannides, "Does Open Source Improve System Security?" *IEEE Software*, vol. 18, no. 5, 2001, pp. 57–61.
7. R. Ford, H. Thompson, and F. Casteron, "Role Comparison Methodology: Measuring Vendors on Customer-Centric Security," Mar. 2005; www.securityinnovation.com/pdf/linux-windows-methodology.pdf.

Jeffrey R. Jones is a self-described "security guy" who has worked in the computer and information security industry for 20 years. He is a director at Microsoft, where he's involved in security improvement efforts under the Trustworthy Computing initiative. His research interests include software vulnerabilities, exploits, and techniques for developing secure code. Jones has a BS in computer and electrical engineering from Purdue University and an MS in computer engineering from the University of Southern California. Contact him at jrjones@microsoft.com.

IEEE Computer Society presents e-learning campus

Further your career or just increase your knowledge

The e-Learning campus provides easy access to online learning materials to IEEE Computer Society members. These resources are either included in your membership or offered at a special discount price to members.



Online Courses

Over 1,300 technical courses available online for Computer Society members.

IEEE Computer Society Digital Library

The Digital Library provides decades of authoritative peer-reviewed research at your fingertips: Have online access to 25 society magazines and transactions, and more than 1,700 selected conference proceedings.

Books/Technical Papers

Members can access over 500 quality online books and technical papers anytime they want them.

IEEE ReadyNotes are guidebooks and tutorials that serve as a quick-start reference for busy computing professionals. They are available as an immediate PDF download.

Certifications

The CSDP (Certified Software Development Professional) is a professional certification meant for experienced software professionals.

Brainbench exams available free for Computer Society members, provide solid measurements of skills commonly requested by employers. Official Brainbench certificates are also available at a discounted price.

<http://computer.org/elearning>