It's hard to compare security alternatives when metrics are nonexistent and data is scarce. But vulnerability and incident reports yield some insights.

Susan C. Lee and Lauren B. Davis

Learning from Experience: Operating System Vulnerability Trends

Reliability engineers can design hardware systems to a specific probability of failure because they have quantitative data on the reliability of the individual parts and a mathematically sound basis for combining part reliabilities into system reliability. Engineers obtain their information on part reliability, expressed as the probability of failure over a given time period or the mean time between failures, through environmentally controlled testing of a large number of identically manufactured parts. Part failure itself is a random process controlled by small, individual variations in the otherwise identical parts.

Security engineers, on the other hand, have no such quantitative basis for designing information systems. Not only is there no known method for deriving an information system's security from knowledge about each of its elements, there is not even data on the security of individual platforms or software elements. And, because software components—the information systems "parts"—have no individual variations, their failure has no statistical element. If a software component fails under certain conditions—for example, unexpected input—all copies of that component will fail under the same conditions. Once a failure mechanism comes to light, software vendors generally fix it, so that discovering one failure mechanism doesn't inform security engineers about future failures. A software security analogy to part reliability is the probability that someone will discover a new, exploitable flaw in a particular component over a given period of time.

Software producers and consumers could conceivably test individual information system components by "red-teaming" them for security flaws and then extrapolate the rate of discovery under test conditions to operational use. But obtaining statistically meaningful results would require tens of thousands of exploit attempts. Because of software's innate complexity, the huge variety of configurations and component combinations, and the frenetic update rate associated with today's information technology, such deliberate testing will always be either inadequate or out of date.

On the other hand, the Internet provides an enormous test bed with a wide range of operational conditions and a large corps of volunteer "testers" (users and hackers). If security engineers could extract statistics from daily experience with networked information systems, they might be able to use probabilistic predictions to design information systems to some desired level of security, much as insurance companies use actu-

Inside

1520-9202/03/$17.00 © 2003 IEEE Published by the IEEE Computer Society

Table 1. Sources of vulnerability data.

Source	Organization type	No. of bulletins
Bugtraq	Service	285
Internet Security Systems	Service	229
Microsoft	Vendor	80
Computer Incident Advisory Capability	Service	65
Caldera Systems	Vendor	63
Red Hat	Vendor	49
Hewlett-Packard	Vendor	33
CERT-CC	Service	30
Sun Microsystems	Vendor	30
NavCIRT	Service	16
Australian Computer Emergency Response Team	Service	9

arial data to set rates that guarantee some probability of profitability. Network managers, in turn, would have some assurance of a defined level of security, and they could choose products according to their organization's needs.

Unfortunately, we don't have the data to support such robust statistical analysis. Privacy and image concerns inhibit data collection efforts by central organizations such as computer emergency response teams (CERTs). The data that is collected supports other endeavors. For example, vulnerability bulletins purposely disseminate just enough information about newly discovered exploitable flaws to let administrators guard against them; they usually don't contain all the information necessary for statistical security analysis. Similarly, the purpose of incident reports is to track and resolve puzzling attacks, not to allow analysis of attack statistics.

Despite their failings, collections of vulnerability bulletins and incident reports still provide the best source for data that could give security engineering a quantitative basis. For this reason, we used vulnerability bulletins to extract statistics about the security of common products—specifically the diverse security characteristics of operating systems in the Windows, Unix, and Linux families. In many cases, the data that was available, rather than security engineering requirements, dictated which statistics we derived. Nevertheless, our findings provide interesting insights into how the various products differ and suggest which security mechanisms would most effectively protect different system designs.

WHERE'S THE DATA?

Our study covered a set of common operating systems and Windows applications. This article discusses only the operating systems: Sun Microsystems' Solaris, Hewlett-Packard's HP-UX, Microsoft Windows NT and Windows 2000, Red Hat Linux, and Caldera OpenLinux. The vulnerability data covered the period from 1 January 1998 through 30 June 2000.

We gleaned our data on product vulnerability from publicly released computer security reports issued by service organizations and individual vendors. This data isn't easy to analyze. The documents are free-form text, so that relevant information might be contained anywhere within a document. Worse, because of the immaturity of the information security discipline, no agreed-upon terminology exists for describing vulnerabilities and exploits, the type of data collected has changed over time, and the many collections of vulnerability and incident data overlap but are individually incomplete.

Service organizations provide thorough, unbiased investigation of exploits as well as some unexploited vulnerabilities or classes of vulnerabilities. Vendor bulletins address vulnerabilities in the company's own products; for example, Microsoft releases bulletins on Windows operating systems. Vendor bulletins typically contain much less technical data on the error itself, but they often address unexploited vulnerabilities that the service organizations might not cover. Table 1 lists the source organizations for our data.

Each of these organizations included different information in its bulletins. CERT-CC geared its reports toward incidents or attacks. Bugtraq focused on vulnerability. Vendors' bulletins centered around patch releases or announcements that a vulnerability reported elsewhere didn't affect their products. All sources included a problem description and a solution (which might be "none"), but not every source covered the platforms affected, the damage that could ensue from exploitation, and the risk of exploitation. Only a handful of sources routinely cross-referenced other sources of information.

This lack of cross-referencing, coupled with the absence of a standard naming system or description terminology, made it difficult to determine whether two bulletins from two sources (or sometimes from the same source) described two different vulnerabilities. In one case, six organizations issued eight separate bulletins for one Microsoft Outlook vulnerability, with the time between the first and last bulletins spanning nearly a year. These bulletins variously described the vulnerability as the Outlook overrun vulnerability, MIME (Multipurpose Internet Mail

Extensions) buffer overflow, MIME name vulnerability, long filename vulnerability, and mail tool vulnerability. Generally, we determined whether bulletins were redundant or unique by studying their full text and comparing their technical details.

One tool we used for reducing redundancy was Mitre's Common Vulnerabilities and Exposures database (http://www.cve.mitre.org). The CVE is a dictionary of vulnerabilities that links vulnerability reports through a common naming system, but it doesn't include technical details or information on risk, impact, and solutions. Still new and evolving, neither the lists of references nor the coverage of vulnerabilities is yet complete. Because CVE excludes technical details, it was sometimes difficult to integrate its references with those not explicitly included in the database.

Despite the difficulty, reducing report redundancy was critical to producing meaningful results. If we couldn't capture all the cross-references to a single vulnerability, whether the bulletins identified it explicitly or not, our data set would contain duplications that would undermine our goal of analyzing distinct vulnerabilities. Ultimately, we identified about 900 security bulletins from our sources pertaining to the common products we were studying, yielding 437 distinct vulnerabilities.

Classifying Security Incidents

To develop the JHU/APL taxonomy, our study's classification system, we studied several existing systems and analyses that used them. The basis for our system was John Howard's work, which we extended to apply to vulnerabilities as well as incidents.

- **"A Common Language for Computer Security Incidents," John D. Howard and Thomas A. Longstaff, Sandia National Lab., 1998.**
- ***An Analysis of Security Incidents on the Internet, 1989-1995,* John D. Howard, Carnegie Mellon Univ., Pittsburgh, Penn., 1997.**
- ***A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems,* Kristopher Kendall, MIT Press, 1999.**
- ***Taxonomy of Computer Intrusions,* Daniel Weber, MIT Press, 1999.**

MAKING DIVERSE SOURCES SPEAK THE SAME LANGUAGE

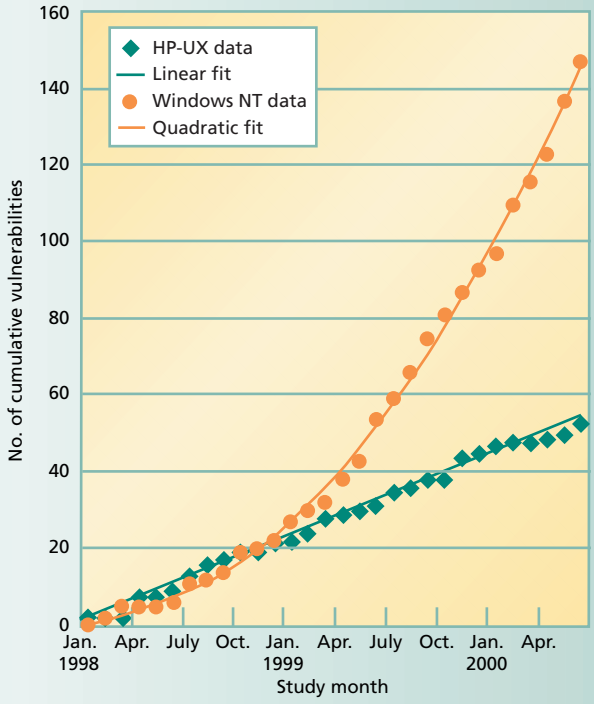
To analyze the data from our widely divergent sources, we first had to develop a terminology and classification system that would let us translate the information into a single language and set of categories. To this end, we developed a taxonomy based on previous work (see the "Classifying Security Incidents" sidebar), but incorporating features we needed for statistical analysis. Besides the products category, which simply identifies the products included in our study, four of the taxonomy categories are relevant to our findings on operating system security differences: vulnerability errors, potential damage, enablers, and corrective actions.

Vulnerability errors

This category refers to a vulnerability's underlying cause. We classified most vulnerabilities as arising from one of the following errors:

- *Boundary condition error,* commonly known as "buffer overflow," results from a failure to properly check the bound sizes of buffers, arrays, strings, and so on.
- *Failure to handle exceptional conditions* covers situations where the system should invoke an exception handler to process erroneous conditions safely but does not.
- *Access validation error* involves insufficient checking of privileges and access permissions.
- *Origin validation error* involves element creation, such as insecure file creation or the improper use of symbolic links.

Figure 1. Vulnerability discovery trend extremes: HP-UX and Windows NT.



- *Input validation error* occurs when parameters and arguments don't fit the program or function specifications, but aren't checked for compliance.
- *Configuration error* denotes an incorrect configuration for the desired or required level of security.
- *Failure to ensure program integrity* results when a program behaves maliciously—for example, for e-mail viruses and worms.

Several other categories of specific errors—race condition, resource, and serialization—covered the few remaining vulnerabilities. For some vulnerabilities, our sources gave no error information. We categorized these as error unknown.

Potential damage

The four common damage categories are disclosure, corruption, theft of service, and denial of service. The JHU/APL taxonomy includes a fifth major category, system compromise, which can include any of the other damage categories.

For example, if a system has sustained an account breakin (at the user or root level), all parts of the system accessible at the privilege level of that user become subject to damage of any kind. We analyzed vulnerabilities that allowed damage at different privilege levels at the highest attainable privilege level.

Enablers

Although enablers are not themselves vulnerabilities, they serve as catalysts by providing the environmental conditions that a vulnerability exploits. Common enablers are

- access to a valid local account, via physical or network access;
- active scripting;
- attachment auto-open or auto-preview;
- file auto-extraction or auto-run;
- default security zones that define permitted activities for various Web site categories;
- file sharing between multiple users; and
- set user ID (setuid) and set group ID (setgid), which define the level (for Unix and Linux) at which a program runs when invoked by any user.

Corrective actions

Although they aren't necessarily solutions, certain corrective actions can reduce or eliminate the risks arising from vulnerabilities. Our taxonomy includes the following corrective actions:

- *fix*—apply patch, upgrade (to new product release), or use temporary workaround;
- *configure*—change settings;
- *disable/block*—turn off a feature;
- *enable*—activate a feature;
- *limit*—restrict access (for example, by egress filtering);
- *prevent*—use external measures such as antivirus software;
- *delete*—remove the problem program, file, or function;
- *relocate*—transfer the problem program, file, or function to another location or another machine;
- *validate;*
- *reboot;*
- *none*—the vulnerability is integral to the OS or application and cannot be corrected; and
- *unknown*—reports mention no corrective action.

RESULTS: OPERATING SYSTEM TRENDS

One of the more interesting results from our study regarded the trends in vulnerability discovery over time. Intuition would suggest that, over time, users would find fewer and fewer vulnerabilities in a particular product as discovery and repair reduced the pool of potentially exploitable flaws. Indeed, this might be the long-term trend if products were used unchanged over very long periods of time. Over our two-and-a-half-year study period, however, quite the opposite proved true.

Figure 1 illustrates the extremes. The HP-UX operating system shows a steady rate of vulnerability discovery over

the study period, with no tendency to diminish over time. In contrast, the number of vulnerabilities discovered in the Windows NT operating system per unit time actually increased significantly over the study period. HP-UX is a relatively stable product with few new-feature introductions. The continuing discovery of new vulnerabilities at a steady rate could simply reflect the size of the initial pool of exploitable flaws; the product might not have been in use long enough to make significant inroads on them.

In contrast, Microsoft briskly updates Windows NT with new features. This continual introduction of features may augment the pool of exploitable flaws without allowing sufficient time for working out older bugs. Table 2 gives the results on vulnerability discovery rates for all the operating systems we studied.

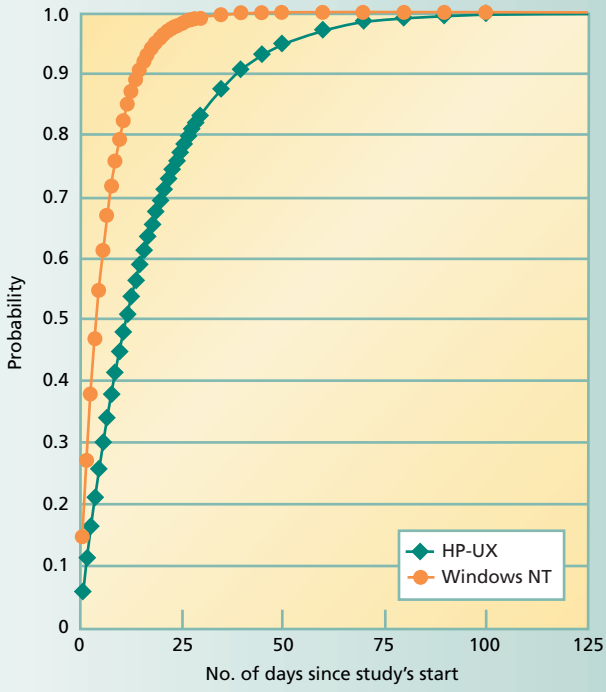
Table 2. Vulnerability discovery trends.

Operating system	Total number of vulnerabilities	Average discovery rate (vulnerabilities per month)	Discovery rate of change (vulnerabilities per month^2)
Solaris	88	2.9	Negligible
HP-UX	52	1.8	Negligible
Red Hat Linux	127	3.8	0.18
Caldera OpenLinux	94	3.1	0.05
Windows NT	143	4.8	0.33
Windows 2000*	39	7.4	Insufficient data

* Windows 2000 was not available until February 2000.

Assuming that we can model vulnerability discovery as a random process (or more precisely, as a Poisson random process), we can use the average rates of vulnerability discovery to predict the probability of the existence of at least one known vulnerability (or any number desired) in these operating systems as a function of time. Figure 2 shows this probability for HP-UX and Windows NT. With better and more refined statistics, you could use information of this sort to select an operating system for its security properties, or to weigh the costs and benefits of an automated push system for vulnerability scan or update alerts.

Figure 2. Probability of at least one known vulnerability.



Operating system profiles

In general, the profiles of vulnerabilities for the different operating systems we analyzed fell into two groups. Unix and Linux are quite similar in the types of vulnerabilities to which they are susceptible. This is not surprising because various Unix and Linux releases include many of the same affected programs. Windows NT and 2000 were similar to each other (of course), but quite distinct from the Unix and Linux group. Although our complete study analyzed each operating system separately, to simplify the presentation here, we've combined results for Windows NT and 2000 into one set and those for Unix and Linux into another set.

We calculated the distribution of error types for the set of vulnerabilities for which our sources specified a particular error as the cause (about 80 percent of the reports). Figure 3 shows this distribution for the two operating system groups. For the Unix and Linux group, a large proportion of vulnerabilities result from boundary condition errors; the error types in the Windows group are more evenly distributed.

We can see one consequence of this error distribution by considering the correlation between error type and damage type. The great majority of vulnerabilities caused by boundary condition errors—82 percent—result in sys-

Figure 3. Operating system error distributions; Unix and Linux operating systems (a) and Windows operating systems (b).

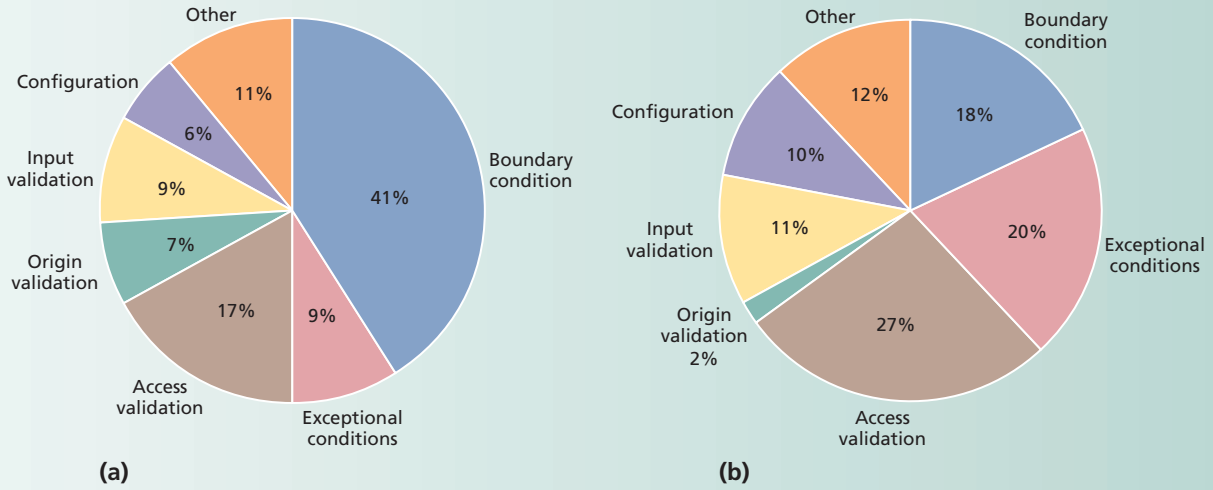


Figure 4. Vulnerabilities sorted by damage type.

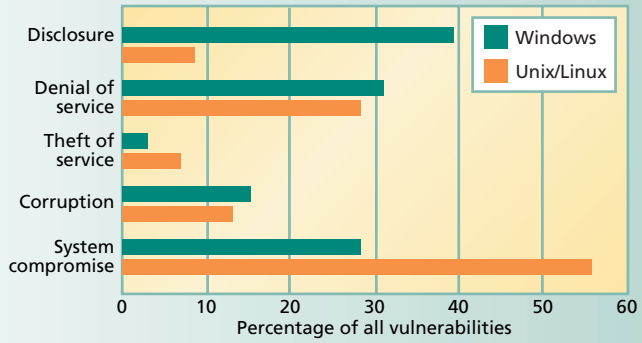
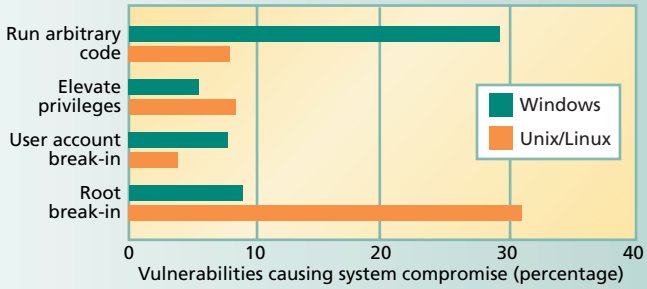


Figure 5. Causes of system compromise.



tem compromise. Because the Unix and Linux operating systems are prone to boundary condition errors, their vulnerabilities are more likely than Windows vulnerabilities to result in system compromise, as Figure 4 shows.

Because system compromise is the most serious damage category, allowing attackers to wreak any type of damage, Figure 5 further characterizes the vulnerabilities resulting in system compromise at the taxonomy's next level. Root break-in is the prevalent damage type arising from system compromise of Unix systems, whereas Windows systems are prone to execution of arbitrary code.

If validated by more and better analysis, our results could provide useful input to the security engineering process. Based on our analysis, a worthwhile investment to guard Unix and Linux systems would be an intrusion detection system that is effective even against novel root break-in attacks, such as the Bottleneck Verification system devised by Lincoln Laboratories (R.P. Lippmann and colleagues, "Using Bottleneck Verification to Find Novel New Attacks with a Low False-Alarm Rate," *Proc. First Int'l Workshop on Recent Advances in Intrusion Detection*, 1998; http://www.raid-symposium.org/raid98/).

For Linux systems, where the source code is available, another possibility is compilation with a tool such as Stackguard (Crispin Cowan and colleagues, "Automatic Detection and Prevention of Buffer-Overflow Attacks," *Proc. Seventh Usenix Security Symp.*, Usenix Assoc., 1998; http://www.usenix.org/publications/library/proceedings/sec98/cowan.

html). Sound investments for Windows systems, on the other hand, would be virus scanning, proxy firewalls, and sandboxing tools.

Controlling vulnerabilities

Some vulnerabilities have what our taxonomy terms enablers, which are not themselves vulnerabilities but allow exploitation of a vulnerability or increase the potential damage. A security engineer cannot control every enabler—access to a valid local account, for example, is not necessarily controllable. Some enablers, however, are configurable settings; security engineers can eliminate many exploits or lessen damage by disabling these enablers. About 25 percent of all Unix and Linux vulnerabilities are contingent on enablers, and 98 percent of those enablers are under the security engineer's control. In Windows operating systems, only 13 percent of the vulnerabilities have enablers, and only 56 percent are controllable.

If a vulnerability has either no enabler or an uncontrollable one, corrective action is necessary. Typically, patches and upgrades are complete solutions for a particular vulnerability; however, 20 percent of all the vulnerabilities we analyzed had neither an accurate patch nor an accurate upgrade available at the time of the vulnerability reports. For 5 percent of the vulnerabilities we culled, the reports state specifically that no corrective actions at all were then available. Not all corrective actions are complete solutions; some just temper the damage or limit use of the software component involved. In cases that require the affected software component's functionality, actions to limit (or eliminate) its use are obviously not viable.

With respect to corrective actions, the Unix- and Linux-type operating systems do not share a profile. As Table 3 shows, the vendors for both Linux implementations we studied almost always offered at least some corrective action. The Unix-type and Windows operating systems failed to offer any corrective action for about 10 percent of the vulnerabilities.

Another difference between Unix- and Linux-type operating systems is the time between a vulnerability's discovery and the patch's availability. (Because companies often don't announce vulnerabilities until they have a fix ready, this figure might not reflect the actual time to patch. However, this caveat applies equally to all the operating systems we analyzed.)

Unix systems have a long delay, possibly because the operating packages include third-party vendor products. Linux turnaround is rapid; because it is open source, the operating system code is universally accessible for modification. Windows turnaround is also quick; this may reflect the fact that Microsoft makes most products included in Windows operating systems, so it can make all necessary modifications in-house.

Table 3. Corrective actions offered.

	Options offered (percentage of cases studied)			
Product	Complete vendor solution	No solution	Work-around	Average time to patch (days)
Solaris	68	6	26	90
HP-UX	85	10	5	55
Red Hat Linux	79	2	19	10
Caldera OpenLinux	94	0	6	11
Windows NT	71	8	21	18
Windows 2000*	60	14	26	14

* Windows 2000 was not available until February 2000.

Our study shows that you can gain useful information by analyzing vulnerability bulletins and incident reports—even considering the limited data they contain, and despite the confounding factors of duplication and nonstandard terminology. The paucity of the data available, however, limits the conclusions that anyone can draw. In fact, the conclusions presented in this paper can be considered tentative at best, because the statistical sample was very small and many confounding factors could not be accounted for. If security engineering is ever to be on a par with other engineering disciplines, the appropriate data will have to be available so that security engineers can derive the relations that quantify and predict performance.

Because controlled testing of systems as complex as commercial operating systems and office applications is impractical, using our everyday experience with networked systems offers the best hope for obtaining the needed data. Today, this opportunity is largely squandered. Creating standards for data collection and reporting, for both incidents and vulnerabilities, would be a first step towards exploiting this data source. Some universal taxonomy—specifying both the type of information that should be collected and the standard terminology that should be used to report it—is essential. In reporting incident data, we must collect information on the environment in which the incident occurred. For example, was the affected system behind a firewall or an intrusion detection device? If so, how were they configured? This kind of information could lead to a quantitative evaluation of these devices' effectiveness. Furthermore, we need some assessment of the damage an incident caused. Understanding the effect of various types of incidents would allow a more cost-effective allocation of defense resources.

Even more importantly, we must collect data in a routine and systematic fashion. Today, CERTs don't collect data on incidents that they choose not to investigate. A study of today's incident data could not even accurately count reported incidents. In addition, incidents are not routinely reported. Many small firms without security teams are actually unaware of penetrations of their networks. Even large companies with considerable IT resources do not always report incidents, fearing that the information may be used to their disadvantage. Accurate statistics cannot be derived from randomly selected bits and pieces of data. To derive lessons from the incidents that occur somewhere every day, industry must put its trust in some central organization or organizations to collect data but report it only in the form of anonymous statistics. Pooling this information in a form that can be analyzed would begin a process leading to future systems that are far more secure. ■

Susan C. Lee *is chief scientist for Infocentric Operations at The Johns Hopkins University Applied Physics Laboratory. Contact her at sue.lee@jhuapl.edu.*

Lauren B. Davis *is a scientist in the Information Assurance group at The Johns Hopkins University Applied Physics Laboratory. Contact her at lauren.davis@jhuapl.edu.*