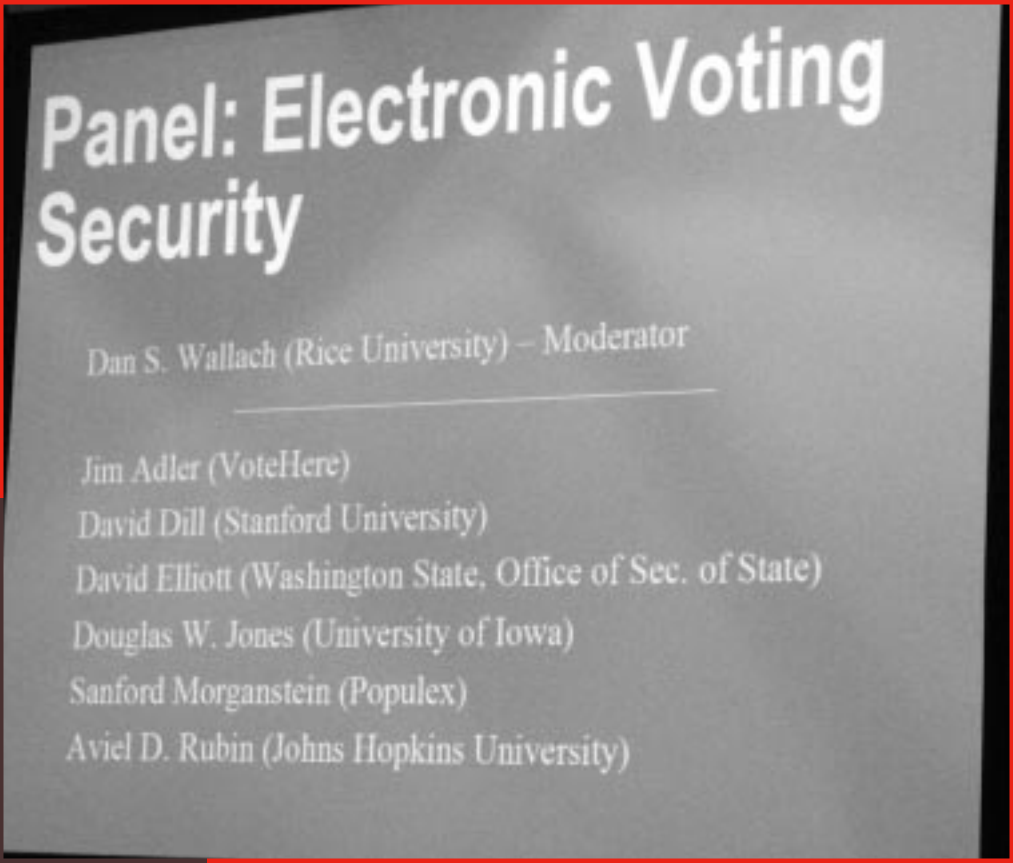
;login:

THE USENIX MAGAZINE

December 2003 • volume 28 • number 6



Panel: Electronic Voting Security

Dan S. Wallach (Rice University) – Moderator

Jim Adler (VoteHere)
David Dill (Stanford University)
David Elliott (Washington State, Office of Sec. of State)
Douglas W. Jones (University of Iowa)
Sanford Morganstein (Populex)
Aviel D. Rubin (Johns Hopkins University)

inside:

Focus Issue: Security

Guest Editor: Rik Farrow

USENIX
The Advanced Computing Systems Association

STOP MONITORING YOUR MONITORS.

THE AGE OF AGENTLESS IS HERE.

10 DAY FREE TRIAL. Chances are your agent-based monitoring system has become a monitored system — monitored by you. It's a relationship you just don't have time for. It's time to go agentless. SiteScope from Mercury Interactive constantly monitors your systems, alerting you the instant a problem pops up. And because it's agentless, maintenance aggravations are kept to a minimum. Download a full version of SiteScope for a free 10 day trial.

www.sitescope.com/trial

MERCURY INTERACTIVE

contents

;login: Vol. 28 #6, December 2003

;login: is the official magazine of the USENIX Association

;login: (ISSN 1044-6397) is published bi-monthly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

$80 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $110 per year.

Periodicals postage paid at Berkeley, CA, and additional offices.

POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

Cover:

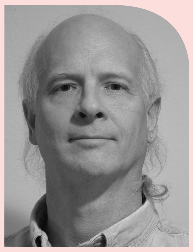
See page 75 for details on this session of the 12th USENIX Security Symposium.

in this issue

by Rik Farrow

Rik Farrow provides UNIX and Internet security consulting and training. He is the author of *UNIX System Security* and *System Administrator's Guide to System V*.

rik@spirit.com



Welcome to the sixth security special edition of *;login:*. Since 1998, I have invited security researchers, lawyers, and other professionals to write articles for these editions. I have also received proposals each year from those who had simply heard about the security edition and wanted to contribute to it. If you want to submit a proposal for the next security issue, May 2004 would not be too early.

I am especially excited about the articles submitted this year. We start off with perhaps the last security paper from Tom Perrine, who worked at San Diego Supercomputer Center until his move to Sony. Tom and Devin Kowatch used the abundant disk and computing capabilities at SDSC to show why crypt-based passwords should never be used anymore. Even without clusters of computers, the time required to crack a single crypt password is too short to believe.

Chris Wysopal writes about why security researchers can find exploitable bugs that software vendors miss. For example, when Microsoft announced the first patch for the RPC vulnerability (MS03-026), security researchers immediately found several other problems, resulting in a second patch from Microsoft.

Jason Damron has researched techniques for application fingerprinting. In his article, Jason shows how various versions of Apache httpd servers can be distinguished just through their responses (ignoring the "Server:" line, if present).

Ed Balas, of Indiana University and the Honeynet Project, explains how the Generation II honeynets can capture encrypted data – unencrypted. His clever ploy uses a kernel module roughly based on the Adore rootkit.

Markus Jakobsson and Filippo Menczer report on a new technique for using email forms for denial of service. Their article includes appropriate remedies that need to be applied to any site offering email forms.

Mudge digs into the realm of tunneling, using protocols in ways they had never been intended for. In particular, Mudge explains how to detect tunneling, focusing on HTTP.

Abe Singer, also of SDSC, writes about how SDSC has survived for many years, very successfully, without firewalls. Abe points out the most important areas for anyone who wants to improve security at a site, especially for sites that demand a large degree of openness.

Renaud Deraison and Ron Gula write about using Nessus. Rather than just offering a rehash of online documents, Renaud and Ron provide hints for making your use of Nessus more efficient, as well as customizing the .nessusrc file.

Dario Forte explains how he and an international team of law enforcement personnel tracked down a slick and aggressive group of computer criminals. According to Dario, this group used custom rootkits to control critical systems, but it was a more common tool that helped to bring the organization down.

Ryan Russell writes about patching Windows systems. Ryan explains why this is one of the most critical security issues that exists today, and why he considers agent technology the right way to go.

Finally, Erin Kenneally has the entire legal component of this issue to herself. Erin has been researching the problems surrounding the use of computer logs as evidence. She includes examples of the use of logs and compares these uses with the current US legal standards for the acceptance of evidence. Erin has also been working with others to create the "secure audit log" model, designed to follow the standards set by the rest of Western law for evidence admissibility.

As always, summaries from the annual Security Symposium are included. As I spent most of the symposium in the basement, running the Ask the Experts track, I personally was very glad to be able to read the summaries, and I want to thank all of the summarizers (again).

I believe that we all enjoyed a wonderful period, roughly coinciding with the growth in popularity of the Internet and going up to the dot-com crash that ended an era of "irrational exuberance," to quote an overquoted Federal Reserve Board chairman. During the Internet runup, technology specialists, including programmers and network and system administrators, were highly valued as the producers of new wealth. Once the bubble popped, the belief in technology as a source of wealth largely vanished as well. Many organizations decided that they no longer needed the services of highly paid specialists. After all, the systems had been installed, and the traffic was flowing over the network, wasn't it?

But we are still in the early stages of computer science and security. Most of our applications have terrible designs, our protocols are flawed, and security is an afterthought. Running networks and systems is still an art, not yet a science. And security? I often believe that we would do better by starting over, with simpler operating systems, cleaner and better documented interfaces, and network protocols based on the lessons we have learned. But how likely is this?

CyberInsecurity

On Thursday, September 25, Dan Geer was fired from his job as CTO of @Stake. Geer is the primary author of the paper "CyberInsecurity: The Cost of Monopoly" (*http://www.ccianet.org/papers/cyberinsecurity.pdf*), in which he is critical of Microsoft, one of @Stake's largest clients. In this paper, co-authored by six other well-known names in security, Geer discusses some issues that are already well known, such as the danger of having a single operating system, a monoculture, that provides fertile ground for worms and viruses. Geer also points out that the "edges" of the Internet are where the fastest growth occurs, with the total computing power of the Internet doubling every 10 months. And the users of these edge systems are not computer security experts. They are home users, and small business owners, who are buying new computers with a Microsoft operating system pre-installed.

Geer also cites the complexity of Microsoft's operating systems. According to Microsoft's own figures, the NT code base has grown at 35% a year, and Internet Explorer at 220% a year. Software experts often describe complexity as proportional to the square of code size. Thus, NT grows by 80% in complexity each year, while IE grows an astounding 380%. Based on the growth in complexity, Microsoft products will have between 15 and 35 times the number of flaws as other operating systems now, with increased disparity in the future.

Microsoft products will have between 15 and 35 times the number of flaws as other operating systems

Geer goes on to write than not all of this complexity is accidental. While Microsoft has begun to employ best programming practices on its internal code, the interfaces provided to application writers are undocumented. Geer uses the proprietary Exchange interface, designed to work with Outlook, as an example.

Microsoft programmers must provide clear modularization and code interfaces on internal code, but the external interfaces are designed to be complex, so that they can-

Signing does not make code secure, but only serves to associate the code to the signer of that code.

not easily be duplicated. The goal of this design is to be anti-competitive. The side effect of this practice means the code that faces the network and all local programs is unnecessarily complex. And that complexity means that there will be more bugs, including many exploitable holes.

Geer et al. do not mention what I consider to be an equally important issue. Back in 1998, when USENIX and Microsoft were co-sponsoring Windows NT conferences in Seattle, Microsoft speakers talked at length about the goal of supporting mobile code. COM was being phased out in favor of DCOM, the Distributed Common Object Model, and would form the basis for future Microsoft operating systems. In this model, code is sent to clients, or servers (SOAP), seamlessly merges with running software, and runs with the privileges of this software. To make this code secure, it would be signed using Authenticode.

Signing does not make code secure, but only serves to associate the code with the signer of that code. In 2002, an excellent example of this appeared when the Microsoft MSN Chat control, an ActiveX object, was discovered to have a buffer overflow vulnerability (*http://www.cert.org/advisories/CA-2002-13.html*). Because the object had been signed by Microsoft, it was trusted by the operating system. And because of the design of Windows, a system without an MSN Chat object could be tricked into downloading the vulnerable object through the use of email containing HTML.

Most Windows viruses spread via email, although other mechanisms, such as Web pages and file shares, are also used. Email- and Web-spread viruses rely on the features of IE to interpret and execute code, making the work of virus writers much easier. Users of IE can deactivate ActiveScripting and prevent the spread of viruses that do not require human interaction, such as SoBig. But doing so also disables all plug-ins. That's right, Flash, sound, the Adobe Acrobat reader, all get disabled as well, to make IE secure (*http://www.spirit.com/Network/ net0502.html* and *http://www.kb.cert.org/vuls/id/25249#solution*).

Ethics

Geer was fired from his position at @Stake the day his paper was published. I found myself defending Geer, whom I know through USENIX, as some people considered what he had done to have been unethical. That is, as an employee of a company which had Microsoft as a frequent client (*http://www.atstake.com/research/reports/eval_ms_ibm/objectives.html*), Geer should not have done anything that would harm the company by which he was employed. He would also have been under NDA (Non-Disclosure Agreements) that might have prevented him from publishing anything harmful about an @Stake client.

I believe that the historical standards for complying with unethical orders come from the Nuremberg trials. During these trials, people who had committed human experimentation and genocide argued that they did not make these decisions themselves, but were acting under orders. The Nuremberg trials produced the standard that a soldier or actor of the state is compelled to disobey unethical orders, even if this will be considered treason.

At this point, you may be wondering what the Holocaust has to do with Microsoft or with Geer's ethics. How can I compare something that involved the torture and death of millions to a computer monopoly, or to the well-being of @Stake and its investors? While certainly not anywhere near the same scale of offense, what Geer did involved a similar ethical decision, one that may impact Geer's ability to get work as an executive in the computer industry. And will the Microsoft monopoly cause millions of deaths? Hardly. But no deaths?

For the sake of discussion, let's consider a specialist in computer security who signs an NDA as part of the process of becoming employed. Note that consultants also sign NDAs before working with a company. Let's present our security person, whom I will call Barnaby, with three different ethical conundrums.

In the first, Barnaby has gone to work for a modest sized, but thriving, software business. As soon as Barnaby gets settled, he discovers that part of his job will be to protect the company's intellectual property. The problem is that this intellectual property was acquired when an executive left another company and brought along the code base from his old company. Should Barnaby comply with the NDA?

Let's raise the stakes. After Barnaby leaves his old employer in disgust, he goes to work for a better-paying job at a pharmaceutical company. Barnaby plows into his work, and it is only after working for months that he discovers the company's big secret. The flagship drug, used for weight loss, has been found to be lethal and is killing, on average, 18 people a year. Should Barnaby comply with his NDA? Keep in mind that disclosing this information would have dire effects on his company's bottom line and on the stock prices so dear to those with options.

In the final example, Barnaby has become an executive in a company that provides code and product reviews for software developers. His company has in the past provided glowing white papers for a software company that has a dominant industry position. This software has become an important element in the monitoring of medical equipment used in ICUs, in nuclear power plants, in the stock market, traffic control, and in many other areas where mistakes can have serious, even deadly, consequences. And Barnaby has discovered that the code in question is nowhere near secure or robust enough to be used in many of the applications where it has been installed. Just the widespread use of this software poses a threat to national security.

Should Barnaby give up his career, possibly violating his NDAs, because he strongly feels that supporting his company's position is not only unethical but may result in death, and certainly will result in considerable economic losses?

Dan Geer is not Barnaby. But in case you believe that this is wholly fiction, remember that the great August 2003 blackout in the US involved a First Energy control facility where the operators lost their consoles during the critical hour when the cascading shutdown occurred – starting on their portion of the grid. Note that this was during the spread of MSBlaster. Also recall that operators of a First Energy nuclear power plant, fortunately shut down for maintenance, lost their consoles when the Windows systems they were using to run X Window servers crashed during the Slammer worm attack. Note that firewalls should have prevented either of these attacks from getting into the networks – but in each case these worms penetrated even supposedly isolated networks.

I find myself standing and applauding Dan Geer for his courage, integrity, and strength. Geer et al. do suggest ways in which Microsoft could modify its strategies to increase the security of all networks. Microsoft would have to give up its monopoly position to do so, and that would be a bad thing for Microsoft, its employees, and its stockholders. But if the alternative is more huge economic hits with each new worm (an estimated $30 billion in August 2003 alone), and potentially the deaths of many (how many people might die during a blackout?), then I think the course that must be taken is crystal clear.

;login:

EDITORIAL STAFF

EDITOR:
Rob Kolstad *kolstad@usenix.org*

CONTRIBUTING EDITOR:
Tina Darmohray *tmd@usenix.org*

MANAGING EDITOR:
Alain Hénon *ah@usenix.org*

COPY EDITOR:
Steve Gilmartin

PROOFREADER:
jel *jel@usenix.org*

TYPESETTER:
Festina Lente

MEMBERSHIP, PUBLICATIONS, AND CONFERENCES
USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Phone: 510 528 8649
FAX: 510 548 5738
Email: *office@usenix.org*
login@usenix.org
conference@usenix.org
WWW: *http://www.usenix.org*

the end of crypt() passwords… please?

by Tom Perrine

Tom Perrine is the Infrastructure Manager for Sony's Playstation Product Development organization, where he watches over online games, software developers, artists, and musicians. In former lives he has been a computer security researcher, system administrator, and operating system developer for universities, companies, and government agencies.

tep@scea.com

Introduction

In the security realm, there is a tendency on the part of of some system administrators and many vendors to ignore "theoretical" vulnerabilities. Inertia is one of the driving forces behind the "full disclosure" movement, a group that believes the only way to get a vendor (or anyone else) to fix something is to release working exploits, so that vulnerabilities cannot be dismissed as theoretical and therefore safe to ignore. It is a recurring issue that it is often difficult to convince vendors (and some system administrators) that vulnerabilities are "real" before exploits appear in running code.

For example, for over 30 years the UNIX password protection system has depended on the UNIX crypt() function to protect 8-character ASCII passwords. Assumptions about computational and storage resources that were made 30 years ago are no longer valid. The UNIX crypt() function has been overtaken by computing technology and should no longer be relied on for password protection.

Although some UNIX and all Linux vendors now offer alternatives, such as longer passwords and stronger hash functions, crypt() passwords are still often required for compatibility with existing account management software or in multi-vendor environments where not all systems support those alternatives. Stronger systems are the norm in the open source operating system community, with Linux and the BSDs all supporting stronger hash functions such as MD5. However, there are still many commercial UNIX variants where crypt() is either the only option or the only option with full support from the vendor, or where using MD5 is incompatible with layered software for that platform.

Until now, quantifying the risks of continuing to depend on this outdated function has been via "back of the envelope" calculations based on key space size and theoretical performance figures for various hardware platforms. The evidence of Moore's Law and academic analysis of cryptographic algorithms have been insufficient to push vendors into supporting stronger systems by default. One of the goals of this project is to drive the last nail into the coffin of the UNIX crypt() function for 8-character passwords, using real-world results. Hopefully, the persuasive power of weaknesses demonstrated by running code and a large database of precomputed password hashes may accomplish that.

Toward this end, over the past few years system administrators and security practitioners at the San Diego Supercomputer Center (SDSC) have investigated the security of the crypt() function in light of ever-increasing computing and storage capabilities.

Our recent paper[1] describes the most recent large-scale password cracking project undertaken at the SDSC. This project examined the use of teraflop computing and petabyte storage capabilities to attack the traditional UNIX crypt() password system.

The paper presents results from applying high-performance computing (HPC) resources such as a parallel supercomputer, abundant disk, and a large tape archive systems, to precompute and store crypt()-based passwords that would be found using

1. Tom Perrine and Devin Kowatch, "Teracrack: Password Cracking Using Teraflop and Petabyte Resources," SDSC, 2003, *http://security.sdsc.edu/publications/teracrack.pdf* and *http://security.sdsc.edu/publications/teracrack.ps.*

common password-cracking tools. Using the Blue Horizon supercomputer at SDSC, we found that precomputing the 207 billion hashes for over 50 million passwords can be done in about 80 minutes. Further, this result shows that for about $10,000 anyone should be able to do the same in a few months using one uniprocessor machine.

This article provides a summary of that work, focusing on the results and implications instead of the technology. Full details of the computing hardware, software, and storage resources are available in the paper.

Project History, Motivation, and Goals

There have been two major password-cracking projects at SDSC. The first, Tablecrack, was intended to quickly determine which UNIX accounts, if any, had passwords that were easily guessed. Tablecrack was used for several years at SDSC to identify vulnerable passwords.

During the use of Tablecrack, it became apparent that there were other interesting questions to investigate, some of which are discussed below. The current project, Teracrack, explores some of these questions, as well as takes advantage of the advances in computing that have occurred since the original Tablecrack project began in December 1997.

Both Tablecrack and Teracrack exploit a novel time/space trade-off to take advantage of very large data storage capabilities, such as multi-terabyte disk systems, on password cracking.

The last and most recent goal of Teracrack was to pursue a "world land-speed record" for password cracking, combining multi-teraflop computing, gigabit networks, and multi-terabyte file systems.

For the purposes of this project, easily guessed passwords are defined as those in a specific list, which is generated using common, publicly available methods (e.g., Alec Muffet's Crack)[2] from publicly available word lists (dictionaries).

The effort in this specific dictionary attack is *not* an exhaustive search of all possible eight-character passwords. For our purposes, it is not necessary to try all passwords, just all those that are likely to be tried by an attacker using commonly available software. In real life, this set of passwords is defined by the software likely to be used by an attacker, e.g., Crack 5.0a or perhaps John the Ripper.[3] By testing the user's password against the set of guesses likely to be tried by this software, we can make a reasonable determination about the user's password falling to an intruder's attack.

The Time/Space Trade-Off

A novel part of both Tablecrack and Teracrack is the reversing of the time/space trade-off. Traditionally, it has been considered infeasible to precalculate (and store) all possible (or reasonable) passwords, forcing the attacker to generate (test) passwords on demand.

In fact, at least one earlier paper on password cracking[4] did discuss implementations of precomputed passwords. However, given the hardware (DEC 3100 CPUs and 8mm tape) of 1989, it took several CPU-weeks to produce hashes on 8mm digital tape for only 107,000 passwords, and it took several hours to check those tapes when searching for hashes. Clearly, it would have been impractical to try to store the hashes for millions of passwords, given disk and tape technologies available at the time.

2. "Crack, a Sensible Password-Checker for UNIX," *http://www.users.dircon.co.uk/~crypto/download/c50-faq.html.*

3. John the Ripper, *http://www.openwall.com/john/.*

4. D.C. Feldmeier and P.R. Karn, "UNIX Password Security — Ten Years Later," Proceedings of Crypto '89, in *Lecture Notes in Computer Science*, vol. 435, pp. 44–63.

It is obvious that the original UNIX crypt() is completely obsolete.

These original assumptions live on in most password-cracking software, including Crack and John the Ripper. These systems assume that there is limited (disk) storage and that each password hash should be calculated from a candidate password, checked against the actual target hash, and then discarded; the computed hashes are not saved for reuse.

Results and Conclusions

First, it is obvious that the original UNIX crypt() is completely obsolete. In today's computing environment, it should certainly not be the default password hash algorithm. It could be strongly argued that the algorithm should not be available at all, that only MD5 or stronger algorithms should be used.

WHAT ARE THE COMPUTATIONAL AND STORAGE REQUIREMENTS FOR SUCH ATTACKS?

We have shown that using the resources present at SDSC it is possible to precompute and save the 207 billion hashes from over 50 million of the most common passwords in about 80 minutes. This means that all hashes for the interesting passwords for a single salt can be computed in about 20 minutes. Further, on a per-CPU basis, the Power3 CPUs used in Blue Horizon are by no means the fastest available. In particular, the Intel Celeron provided very good numbers in our crypt timings. If, as we believe, the numbers above are related to the actual performance of Teracrack, then a modern x86 (1-2GHz) should be able to hash our word list for one salt in far less than 20 minutes.[5]

5. A test run indicates that this is the case.

To save 207 billion hashes requires 1.5 terabytes of storage. We had this much storage in a network file system, connected over high-speed network links. We also have a multi-petabyte tape archive to provide long-term storage for the hashes. The I/O bandwidth required for a full 128-node run is an average of 80 megabytes per second; however, in that time a single process only averages 80-82 kilobytes per second. Further, running the post-processing requires 2.27 terabytes of storage, and the resultant .pwh files will require 2.26 terabytes. Note that this space is not cumulative, but just represents different amounts at different times as the work is done.

These requirements are probably out of the reach of typical machines used by a single attacker. We were trying to exploit the resources available to us, and thus were trying to make this run in as little real time as we could. There are other methods we could have used that would have been less demanding. A more patient attacker could use fewer resources and still launch a successful attack in a reasonable amount of time, such as a few weeks.

ARE LARGE-SCALE DICTIONARY ATTACKS FEASIBLE FOR ATTACKERS WITHOUT ACCESS TO HIGH-PERFORMANCE COMPUTING RESOURCES? IS A DISTRIBUTED (COOPERATIVE) EFFORT FEASIBLE?

The computational requirements for precomputing all the hashes are high, but not prohibitively so. For a single Power3 CPU, the bulk encryption phase should take about 60 days. It should take a single CPU on a Sun-Fire 15K about 13 days to do the post-processing. Ignoring the storage and I/O bandwidth issue, the time required for bulk encryption and post-processing phases should decrease linearly for every CPU added to it.

The storage requirements are also "reasonable." With the current availability of IDE disk drives larger than 200GB, perhaps in IDE RAID arrays, the storage requirements are also accessible. In fact, SDSC has experimented with low-cost "terabyte-class" file servers.[6] We have recently (November and December 2002) purchased 1.2-terabyte PC-based RAID file servers for around $5,000. We have been quoted prices below $10,000 for 2.8-terabyte file servers.

6. SDSC, "A Low-Cost Terabyte File Server," *https://staff.sdsc.edu/its/terafile/*.

The I/O bandwidth should not be a problem for the smaller machines which are more likely to be used by an attacker. Even a dual-processor 1.5GHz x86 machine would generate less than 2MBps of output. If the machine only writes to local disk, there will be no network-bandwidth problems.

There are also several strategies for coping with the total amount of storage space required. Most involve either distributing the computation or making a space/time trade-off.

- By distributing the computation across several machines, the storage space required on each machine would be greatly reduced. It would also allow using existing hardware, with the addition of a large IDE hard drive. For example, a cooperative effort with eight machines could add a 200GB IDE hard drive to each machine. This would provide 1.6TB of storage, which is enough to store the output of the bulk encryption phase. Using the next technique, it will be enough to handle the post-processing phase as well.
- The post-processed output requires 50% more storage space than the raw hashes. This is only because the output also contains a reverse pointer that allows retrieving the password as part of the table lookup. Having this pointer is not necessary, even for an attacker who is trying to recover the password. The post-processing is still needed to sort the hashes (and speed lookup times), but the final space requirement will be 1.5TB instead of 2.26TB if the reverse pointer is left out. Instead of using the precomputed hashes to retrieve passwords, the attacker can use them to figure out which passwords can be recovered with minimal effort. Here the attacker uses the precomputed hashes on a stolen password file to determine which hashes are in the attacker's dictionary. Once it has been determined which passwords will be found, it will take 20 minutes or less per salt to recover the password. As it only takes one password to compromise an account, a single 20-minute run should suffice. Using this method, however, an attacker can still recover over 30 passwords in a single day. The savings is from not wasting time attacking strong passwords.
- We did not investigate compression at all. However, in Tablecrack, it was found that an algorithm like the one used by Crack to compress dictionaries showed promise.
- While over a terabyte of disk storage is still expensive, 200GB is very affordable, and enough space to store precomputed hashes for 400 salts. While having 400 salts is not as good as having all of them, all it takes is one broken password.

Thus, it seems safe to say that large-scale dictionary attacks are feasible for either a very patient single attacker, an attacker with a farm of compromised machines, or a collective of cooperating attackers. What we were able to do in hours, a network of attackers could easily do in days.

ARE THERE COLLISIONS (MULTIPLE PASSWORDS THAT PRODUCE THE SAME HASH) IN THE PASSWORD SPACE?

We found two types of collisions. First, there were some words in our dictionary which contained characters with the high-bit set. There were 24 such collisions in each case

Even a small set of . . . attackers, or an attacker with moderate resources, could independently duplicate our work.

the two colliding words only differed in one character. Also in each case the differing characters were the same in the lower seven bits. These collisions are due to the way crypt() makes a key from the password, by stripping off the high bit, and concatenating the lower seven bits of each byte to form a 56-bit key. The lesson from these collisions is that there is no benefit from including characters which use the high bit in a user password, even if your version of UNIX supports this.

Second, we found one "real" collision. By this we mean two words that differ in more than just the lower seven bits of each byte, which hash to the same value. This occurs with the words $C4U1N3R and SEEKETH, under the salt 1/. Both words hash to ChERhgHoo1o. The lesson from this is that although there are collisions in the crypt algorithm, and they do reduce the usable key space, they are relatively rare and this is likely not a real-world concern. Quantifying the exact number of collisions would require a dictionary equal to the key size.

WHAT ARE THE NONTECHNICAL (SOCIAL, ETHICAL, AND LEGAL) ISSUES INVOLVED IN MAKING THE RESULTS OF THIS PROJECT PUBLICLY AVAILABLE?

This question has been at the heart of both the Tablecrack and Teracrack projects and was not adequately addressed by either.

With Tablecrack we examined the issue and made a decision to only store password hashes, and not include the "back pointers" to the original passwords. This decision was mostly to address storage concerns, but also made it at least slightly more difficult for attackers, even if they had access to our password hashes. At that time, we considered that the most likely misuse of the Tablecrack data would be by an insider, due to the difficulty in moving the large data sets outside of SDSC. We expected that even if the attacker could determine sets of crackable passwords, we would have a good chance of detecting the resulting attacks on the identified vulnerable passwords.

While designing Teracrack, we realized that data storage and CPU performance had advanced to the point that even a small set of cooperating attackers, or an attacker with moderate resources, could independently duplicate our work in days or weeks. This changes the issues considerably.

We have investigated several ways to make our results available for system administrators to check their own password hashes for weak passwords. None of the ways is completely satisfactory, for various reasons:

1. We started by looking at providing a Web interface, such as a search engine: submit a Web form with a UNIX password hash and we could tell you whether or not the hash was from a weak password. This has problems in terms of back-end lookup performance, online storage, and our complete inability to prevent this system from being abused by an attacker. Such an interface, if it existed, would quickly succumb to the dreaded "slashdot effect" and become useless.
2. It was suggested that we could improve the Web interface idea by occasionally returning "weak" for a strong password. This would cause any attacker to occasionally waste CPU time trying to crack an "un-crackable" password. For the legitimate user, we would occasionally influence them to change a password that was not weak. It can be argued that it is never a bad idea to influence a user to change a password, but this is only true if they don't replace a strong password with a weak one.
3. We then decided that the problem was authenticating the user of any system we might build. We decided that we could probably find a way to manually vet users,

registering people whom we could identify and decide we trusted as users of our system. Although we could generate a list of well-known individuals, and individuals who were personally known to us, this obviously does not scale. If this is still accessed via a Web server, the problem of ad hoc query performance remains.

4. Our last thought experiment combines the ideas of user registration and bulk lookups. We could register the PGP keys of people we trust not to abuse the system. These people could send formatted email messages, signed with the registered keys. We could batch all the requests from all the users in each 24-hour period into a subset of lookups. This would have several advantages. It would allow us to batch queries by salt, so that we would have to make only one pass through each salt's file. Additionally, if the hash files were stored in HPSS, we would only need to retrieve the files of the salts that were in at least one query. With fewer than a thousand or so queries in a batch, it is likely that we would not need to retrieve more than half of the per-salt files.

Unfortunately, this system still suffers from problems of scale in handling user subscriptions, problems of policy in determining who may use the system, and the cost of actually operating such a system.

In the end, it is not clear how we can make the resulting data publicly accessible. Even if we can satisfy our own concerns, there would be liability issues if it could be shown that our system was used by an attacker to mount a successful attack against someone's weak passwords.

Yet all of this may be moot, as we have shown that this work can be recreated by a determined, patient attacker.

For ourselves, using a batch mechanism to submit the information from SDSC's password files will allow us to find weak passwords in a timely fashion, until we have completely eliminated the use of crypt() passwords on all our systems.

Future Work

There are six main areas in which we would like to pursue this project further:

1. Performance tuning for better scalability. We would like to attempt a few methods for reducing or eliminating the runtime connection to the number of nodes. Also, there are performance tweaks which show promise but were nonfunctional at publication time. Some of these include asynchronous I/O and dividing the word list rather than the salts.
2. Moving the software to emerging hardware platforms. SDSC is currently installing a new system that may offer up to 34 teraflops, with scaler integer performance at least 30 times that of Blue Horizon. This system will have very different cache, main memory, and network and I/O performance.
3. Public access to check for weak passwords. We would like to allow subscribers to check their site's passwords against our precomputed hashes. Thus subscribers could verify that they have no passwords in the Crack dictionary, without needing to invest in the resources to run Crack. Even though we have identified some of the problems above, there should be some way to make this service available for legitimate use.
4. Measurement of different-sized word lists. We would like to try measuring runtimes for word lists which are both larger and smaller. The larger word list would

likely come from adding foreign-language dictionaries to the initial dictionary. The smaller word lists would simply be subsets of our current word list.

5. Algorithms other than the traditional DES-based UNIX `crypt()`. We would like to try precomputing an effective number of passwords for other algorithms, including SHA-1 and MD5. Precomputing the Microsoft "LANMAN" hashes would be particularly easy, as the passwords are limited to uppercase and the hash is not salted. This would effectively be a massively parallel "L0phtCrack."
6. Investigating `crypt()` performance on x86. The x86 architecture seems to run `crypt()` very quickly, but just how quickly depends on a variety of factors. We would like to examine factors such as cache and CPU core versions. However, since we are arguing that `crypt()` should be eliminated, we should actually focus on the performance of MD5-based hashes instead.

Acknowledgments

This article is based on the Teracrack paper, which would not exist without the significant Teracrack development work by Devin Kowatch, and the contributions by the system administration staff at SDSC, especially Jeff Makey, the creator of "Tablecrack."

Availability

The code used for this paper is publicly available at *http://security.sdsc.edu/software/teracrack*. It is covered under the U.C. Software License, which allows source code access and is free for noncommercial use.

learning security QA from the vulnerability researchers

by Chris Wysopal

Chris Wysopal is the vice president of research and development at @stake, where he leads research on how to build and test software for security vulnerabilities.

cwysopal@stake.com

Every day, vulnerability researchers find and publicly disclose new vulnerabilities for software products. Many of these products are made by vendors who assure us that they know how to create secure software. What makes it possible for a vulnerability researcher, who usually doesn't have access to design documentation or source code, to find these problems? He would seem to be at a major disadvantage compared to the vendor's testing team. Is the vendor not looking? Or is there something about the process of discovering vulnerabilities that keeps software vendors from doing it well? This article will take a look at the differences between researcher and vendor and how these differences lead to different results.

First, let's examine a bit of history. The setting is the 1997 USENIX security conference. Mudge from the L0pht is hanging out with Hobbit, who, while not officially part of the group, often collaborated with it. The two are approached by Paul Leach, a Microsoft security architect, and other senior technical people from Microsoft. The gentlemen from Microsoft wanted to sit down and learn how Hobbit had discovered vulnerabilities in the Windows CIFS protocol[1] and how Mudge managed to find flaws in Windows NT's network authentication. Over a fine dinner and a few bottles of wine, the two researchers took the Microsoft security folks through the process of black box reverse engineering, with a vulnerability twist. This is what they described.

1. Hobbit, "CIFS: Common Insecurities Fail Scrutiny," 1997, *http://downloads.securityfocus.com/library/cifs.txt.*

The first task for attacking the CIFS protocol was to install a host with a sniffer on the network between two hosts running Windows. The sniffer was running on a non-Microsoft OS. The reason for this wasn't just because Hobbit's coding skills happened to be better on UNIX. It was also to make sure that the analysis host's OS wasn't contaminated with any Windows internals knowledge. If a Windows host was used for analyzing the network traffic, the network stack or other internal OS components might perform hidden data manipulation. This is a theme that will pervade the reverse engineering process. All analysis tools, whether off-the-shelf or custom coded, need to be free of unwanted interaction with the program under test. For network analysis, a different OS often provides enough isolation. For analysis programs that must run on the same host, it is best to avoid using higher-level OS APIs that might perform unwanted data manipulation.

With the sniffer in place, CIFS transactions were performed between the two Windows hosts, and the network packets were recorded. Transactions were repeated with slight changes, and the differences in the packets were noted. This was a laborious process, but over time it was clear what different packet types there were and what data fields were in these packets. The gentlemen from Microsoft were surprised by the approach. It had never occurred to them to analyze the protocol this way. After all, they had Windows API functions they could call to look at the data in the CIFS transactions. They had design documentation to tell them what was in a particular field within a packet. Their analysis approach differed in that they were seeing how the CIFS protocol was supposed to work, not how it actually did work. With his independent-analysis approach, Hobbit was able to discover workings of the CIFS protocol that were unknown to its designers and implementers.

It is hard to believe in this age of heightened security awareness that most people who develop software still don't know how vulnerability researchers work.

What may seem like liabilities on the vulnerability researchers' side – using a different OS, having no design documentation or code to look at, and having no access to internal testing tools (which may share code with the system under test) – are turned into benefits. The researcher is not tempted to take a time-saving shortcut while analyzing the system. He must build up from scratch what the bits on the wire mean and how they can be changed. He gets an unbiased view of how the program actually works.

It is understandable that the security folks from Microsoft didn't know how vulnerability researchers worked in 1997. Vulnerability researchers were a small, closed group of people dealing with something fairly arcane. Today, software security affects every computer user, from those in the military and government to the teenager at home. It is hard to believe in this age of heightened security awareness that most people who develop software still don't know how vulnerability researchers work. The software-testing community needs to learn why these researchers are successful and start to work like them, though perhaps a bit more formally. Otherwise, vulnerabilities that could have been found before a product is delivered to the customer will be found by researchers and end up needing to be patched, or worse, be exploited.

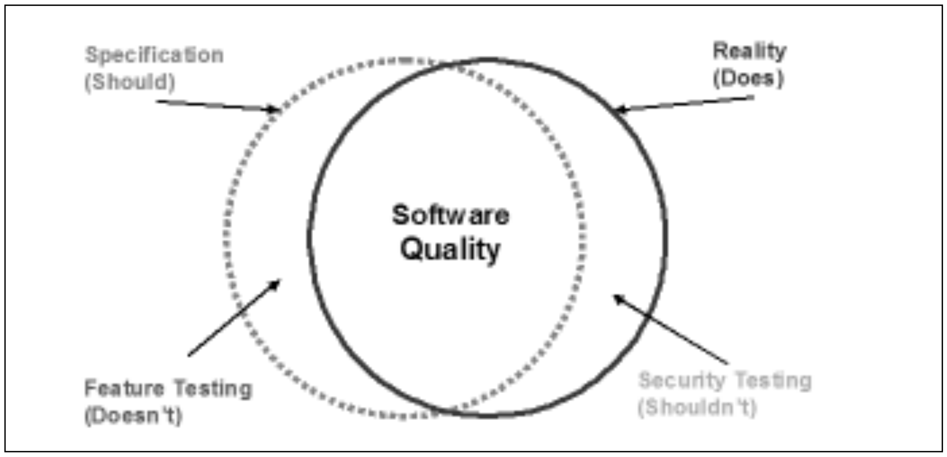
The first thing the researcher does is zero in on the weakest links in the software, the areas of highest risk. Ad hoc threat modeling is performed on the data as it flows in and out of the program. Where can the attacker inject data into the program? Where are the places that data can be injected without first performing authentication? This is the primary attack surface.

Many software testers, when they actually take time to do security testing, get bogged down looking at the security throughout the entire application. This is understandable, since they are used to testing the functionality of the whole application. But security testing is very different from feature testing. When there is limited time, and there always is, there is a need to start at the areas of highest risk and continue toward areas of lower risk.

The other major shift that testers need to make is to stop thinking of security as a feature that can be tested with positive functionality testing. Positive testing is making sure a feature works. If the program has authentication and access control lists, those are typically tested. Vulnerability researchers almost never look at the security features. Positive testing will not find out that a program contains a stack buffer overflow in code that reads data from the network. Testers need to learn the art of negative testing, the art of causing faults.

In the perfect development world, a finished program would exactly match the functionality of its design specification, with no more functionality and no less. In the graphic on the facing page there are two circles, one representing the program's design and the other representing the actual implementation. Since we have not perfected software coding, there is a need for testers to find the mistakes that coders make. These are the mistakes that lead to the design not matching the implementation. But most testers only cover the deviation, where the implementation is lacking functionality defined in the design. What about the deviation where the program has functionality that was unintended? This is where the vulnerability researcher's skills come into play. This is where knowing what a program actually does and not just what it was designed to do is critical.

Negative testing is identifying the inputs of the program and putting in data that is obviously invalid. Typically, the data is nowhere close to what a normal user would do.



Design vs. Reality[2]

Many testers can't imaging anyone inputting a username of 50,000 characters, so they don't make it part of their test plan. But attackers do try such seemingly ridiculous inputs, so the negative testing plan should too.

Luckily the tester does not need to input this manually. State-of-the-art vulnerability research involves automated fuzzers that can perform the fault-injection process. Fuzzers have a list of rules for creating input that is known to cause errors in processing: long strings, Unicode strings, script interpreter commands and delimiters, printf-style format strings, file names, etc. @stake's WebProxy tool does this for HTTP. Immunity Security's SPIKE is a fuzzer-creation toolkit that can be used to fuzz arbitrary network protocols. Once you have a fuzzer for the protocol you want to test, you just run it against a debug version of the software running in the debugger. Chances are the program will eventually crash and you will be sitting at the vulnerable line of code in the debugger.

There is much more for the software tester to learn, but a great start is to learn how to threat model for the highest-risk attack surfaces, understand negative testing, and get up to speed on fuzzing. Follow the online security community to learn the tools and techniques that vulnerability researchers use and make them part of your quality assurance process. Every vulnerability found during QA is one less for the vulnerability researchers to find and one less vulnerability for software users to patch.

2. H. Thompson and J.A. Whittaker, "Testing for Software Security," *Dr. Dobb's Journal* (November 2002).

identifiable fingerprints in network applications

by Jason Damron

Jason Damron has been involved in network security for 10 years. He is currently the lead developer of the Dragon Network Intrusion Detection System for Enterasys Networks.

jdamron@enterasys.com

It is important for both an attacker and a defender to know exactly what is running on a target system. Knowing not only the specific application but also the version number allows one to enumerate the vulnerabilities that are present. Application fingerprinting is a method of determining the type and version of an active network server or client. This is an advanced technique that replaces banner grabbing for situations where a banner doesn't exist or has been removed or obscured.

Typically, banner grabbing consists of initiating a connection to a network server and recording the data that is returned at the beginning of the session. This information can specify the name of the application, version number, and even the operating system that is running the server. Other protocols require that a request first be made, with the resulting response holding the server information.

Savvy administrators have started removing or obscuring this data, making banner grabbing useless or misleading. For instance, an Apache server can be modified to respond with the banner of a Stronghold, an ISS/5.0, or a completely contrived server. Adversaries may then unknowingly choose attacks that will not affect the real Apache server. This method of security through obscurity (while not a good practice by itself) can be a profound gain, since the time it takes to obscure is far less than the time and effort it can take to overcome it.

Application fingerprinting is the field of study that can be used to overcome ambiguity and misdirection. It can be used actively or passively to detect fine distinctions in network programs that give away specific product and version information. Because of this, security through obscurity can definitely increase the challenge but cannot completely remove the threat of a determined attacker's ability to identify the running service. Some have started to undertake the task to counter this "raising of the bar" and are providing tools such as amap and vmap.[1]

1. "The Hacker's Choice – Releases," *http://www.thc.org/releases.php*.

What to Fingerprint?

Any active service on a network can be fingerprinted to some extent. Passively watching the traffic flow by is one method of fingerprinting, but it relies on fingerprints found in normal traffic. This is most efficient when banners are present within the information transfer. However, when banners are not present this method can lose granularity, and it may take a large volume of traffic for the different variations required to precisely determine the application and version information. Active approaches yield much faster results, since the traffic can be intelligently chosen in order to focus the identification process.

Client software can also be fingerprinted. This can be important knowledge in preventing a network breach due to a client-side security flaw. Rogue Internet servers could be loaded with potentially damaging payloads for vulnerable Web browsers, chat clients, media players, or other client programs. These rogue servers could potentially force the traffic of the conversation in a way that will uniquely identify the client. At that point, it could respond with malicious content targeted to the specific client, producing a higher probability of success.

Fingerprinting unknown ports may be somewhat more complicated, because the protocol must be determined first. Banner grabbing can start with the initial return, but if the banner is obscured, deeper inspection must occur. This process may break down to simple trial and error, cycling through the protocols that have been researched.

Even typically passive devices such as an Intrusion Detection System could be identified if set up for active response. An attacker could force traffic to be emitted, triggering a defensive measure such as a TCP session termination. Then by examining the returning traffic for the number of TCP RST packets sent, how fast the sequence numbers grow in attempting to catch up to the ongoing traffic, static values such as IP identity field, and other possible indicators, the fingerprinting software can determine the make and model of the IDS.

Intrusion detection systems have started applying banner grabbing and modeling technologies.

How Is It Possible?

Application fingerprinting works because all actively developed software evolves. New software versions typically include some combination of bug fixes, updates, rewrites, and completely new features. If any of these affect its network presence, it allows the software to be uniquely identified from its predecessors.

The fact that some applications can be uniquely identified by a remote agent is not a vulnerability in the software. In fact, it can be viewed as the opposite, in that vulnerabilities are being removed, which can change the network's characteristics. Ideally, developers who become conscious of this will choose to create a normalized network interface. They could allow modified interactions to take place only when needed by new functionality configured to be active by the user. Alternatively, they could allow configurations that removed unneeded functionality to retain the conventional interaction.

Intrusion Detection

Intrusion detection systems have started applying banner grabbing and modeling technologies. An example of this would be picking out the Sendmail banner and remembering what it is for future reporting, or generating an event immediately if it is known to be vulnerable.

An implementation of application fingerprinting could be the identification of a DNS server from observing a normal DNS conversation. Typical DNS traffic does not include any version information to trigger current systems. However, with careful research it may be possible to detect nuances of the conversation that give away the type and version of the DNS server. At this point, the IDS can respond like any other fingerprinting system – by learning and applying that knowledge to present more intelligent alerts.

Another example could be the security administrator of a large enterprise who doesn't own or operate all of the Web servers. If one of his users makes the effort to obscure the server type and version, they have also obscured it from the IDS. However, if the IDS can monitor enough conversations to determine the type and version, it can then generate events with better information about whether the server was susceptible to the attack.

The above examples are narrow in scope, since they assume a standard port or that the underlying protocol is at least known. A broader version of this idea is an application fingerprinting engine. This engine could be fed by any unencrypted network traffic and first determine the protocol, then move to the type of server/client, and finally, identify the version (if possible). For instance, if a user decided to hide and obscure a Web server on port 44322, an engine such as the one described above could attempt to determine all of the server characteristics to enable the enhanced reporting available for that application.

Case Study: Apache 1.3.x

WHY APACHE?

Apache is the most widely utilized Web server in the world.[2] It is also open source, so code for every version can be downloaded and checked for differences. Previous versions are available at *http://archive.apache.org/dist/httpd/old/*. Other good choices to examine for fingerprints would be BIND and Sendmail. Both are used extensively, and source code for the current and prior versions is freely available. Also, both BIND[3] and Sendmail[4] have had security-related issues in the past, which makes it more important to be able to identify any non-current servers.

Please note that this is not a vulnerability of the Apache Web server. As Apache grows and evolves there will be changes, and some will cause its network presence to change.

2. "Netcraft: Web Server Survey Archives," *http://news.netcraft.com/archives/Web_server_survey.html*.

3. Internet Software Consortium: BIND Vulnerabilities, *http://www.isc.org/products/BIND/bind-security.html*.

4. Vulnerabilities found by searching SecurityFocus vulnerability archive for "Sendmail Consortium," *http://www.securityfocus.com/bid/{2794, 2897, 3377, 3378, 4822, 3163, 5770, 5921, 6548, 5845, 5122, 7614, 7829, 6991, 7230, 8485}*.

TEST SETUP

For this case study, we will assume default installations of Apache servers. The only changes made to the configuration were to make each version listen on a different port for concurrent testing and to start a single instance of the server, since performance will not be an issue. Also, the scope of this research was limited to UNIX versions of the server.

WHERE TO START

The safest and most direct method is to make simple requests. A query for the DocumentRoot can provide useful information such as the options that are present, the order of the options, syntax of the options, and maybe the default Web page, all of which can be used to identify the server. This information is more likely to help determine the type of server rather than an exact version.

Next, simple request errors can be made, such as requesting pages that are not present or leaving out required headers (HTTP/1.1 requires that the Host: header be present). However, because not all Web servers are as robust as Apache, these types of attempts to fingerprint may cause poorly implemented servers to crash.

The accompanying matrix illustrates three simple requests that can be used to narrow down the version of Apache that is being run. They are listed in order of obtrusiveness, starting with the safest method.

Apache Version	TCN Present	Negotiation Options	Bad Method (HEAD 1)
1.3.27	X	Fewer than 24 (missing .lu)	Error 1, 4, 5
1.3.22		Similar to 23	Error 2, 4, 5
1.3 {12, 14, 17, 19, 20, 23, 24, 27}	X	Only 17 and 19 match	Error 1, 4, 5
1.3.11	X	Fewest options	Error 1, 4, 5
1.3 {4, 6, 9}		None	Error 1, 4, 5
1.3.3		None	Error 1, 5
1.3.2		None	Error 3, 5
1.3 {0, 1}		None	Error 1

Returning Error includes:

1. *HEAD1 to /index.html not supported.<P>*
2. *Same as above except URL reads /index.html.en*
3. *Invalid method in request HEAD1 / HTTP/1.1<HR>*
4. *Same as above but ends with <P>\n<HR>*
5. *<ADDRESS> tag present*

As can be seen , several of the unique characteristics of a default Apache installation revolve around its ability to negotiate. The first test uses the existence of the Transparent Content Negotiation (TCN) header in any response to divide up the versions. Versions 1.3.11 to 1.3.27 can be broken down further by submitting Accept: statements with an invalid type to force the server to present all known content types. Over time, the supported content types have changed in the default configuration and this can be used to determine the specific versions. The older versions can be broken down further by submitting improper requests and checking for uniqueness in the results. Some of the releases, such as 1.3.{4,6,9}, have very few functional changes in the protocol code, which makes version distinction very difficult. In those cases, fingerprinting may become dependent on the additional modules that have been loaded.

The first step in disguising an application is to enable only the functionality that you require.

RAISING THE BAR HIGHER

The first step in disguising an application is to enable only the functionality you require. The more functionality that is utilized by the network server, the greater the chance for identifying anomalies to be present. Likewise, you don't want to expose yourself to potential security risks that may exist in unnecessary functionality.

The next step is to create a custom configuration to remove as many default responses as possible. Some software does not provide enough flexibility to remove any standard replies. In these cases, either source modifications or binary editing would be required to create the necessary obfuscation. The following are some examples of the options Apache administrators have to increase the complexity required to determine the version information.

Starting with the 1.3.x tree, Apache introduced the ServerTokens directive, which allows the administrator to manipulate the responding Server: header by applying it to the httpd.conf file. However, the version information was always present until version 1.3.12, when it allowed the Prod[uct Only] option, which limits the display to just "Apache."

To further obfuscate the Server: response header, a simple change can be made in the source code. The include file {ApacheTree}/src/include/httpd.h contains #define statements for the product information. The product name can be changed to read:

```
#define SERVER_BASEPRODUCT "GuessMe"
```

Then setting the ServerToken directive to Prod[uct Only] will cause Apache to answer with the following "Server" header:

```
Server: GuessMe
```

The ErrorDocument directive can also be used to deter inquisitive minds from uncovering the type and version of the server. This option, when placed within the httpd.conf file, allows Apache to serve up custom error pages which will remove some of the low-hanging fruit of fingerprinting.

The following is an example of using ErrorDocument to remove identifiers:

```
ErrorDocument 500 /standard-error.html
ErrorDocument 404 /standard-error.html
```

The negotiation alternatives of an Apache server can be modified in the httpd.conf file. When the mod_mime module is enabled, only the languages and character sets that need to be supported should be included, using the AddLanguage and AddCharset directives. Also, if the mod_negotiation module is enabled, the language priority list should also be altered to reflect only languages that are required. The following is a modified section of the httpd.conf file that limits the language priority to only English (en) and German (de):

```
<IfModule mod_negotiation.c>
    #LanguagePriority en da nl et fr de el it ja kr no pl pt pt-br ru ltz ca es sv tw
    LanguagePriority en de
</IfModule>
```

Following the procedures specified above negates many of the fingerprinting techniques discussed and takes a surprisingly small amount of time. Without making these changes, creating fingerprints for individual versions of Apache is a time-consuming task. Spending the extra time to customize the configuration can remove unique characteristics, which causes the application fingerprinting to become much more difficult.

Statement of Ownership, Management, and Circulation, 10/01/03

Title: ;login: Pub. No. 0008-334. Frequency: Bimonthly. Subscription price $110 individuals and institutions. Office of publication: USENIX Association, 2560 9th Street, Suite 215, Berkeley, Alameda County, CA 94710. Headquarters of Publication: Same. Publisher: USENIX Association, 2560 9th Street, Suite 215, Berkeley, CA 94710. Editor: Rob Kolstad. Managing Editor: Alain Hénon, all located at office of publication. Owner: USENIX Association. The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes has not changed during the preceding 12 months.

Extent and nature of circulation	Average no. of copies of each issue in preceding 12 months	Actual no. of copies of single issue published nearest to filing
A. Total no. of copies	8,068	7,550
B. Paid and/or Requested Circulation		
Outside-County Mail Subscriptions	5,223	5,111
In-County Subscriptions	148	0
Sales through Dealers	2,179	1,945
Other Classes	0	0
C. Total Paid and/or Requested Circulation	7,550	7,056
D. Free Distribution by Mail		
Outside County	0	0
In-County	0	0
Other Classes	32	33
E. Free Distribution Outside the Mail	333	400
F. Total Free Distribution	365	432
G. Total Distribution	7,915	7,488
H. Copies Not Distributed	199	515
I. Total	8,068	7,550
Percent Paid and/or Requested Circulation	94%	93%

I certify that the statements made by me above are correct and complete.
Alain Hénon, Managing Editor

sebek

Covert Glass-Box Host Analysis

by Edward Balas

As a network security researcher at Indiana University's Advanced Network Management Lab and Honeynet Project team member, Edward Balas has focused on network infrastructure protection. Edward's professional interests include Network Monitoring and Traffic Analysis, as well as advanced honeynet data capture techniques.

ebalas@iu.edu

Introduction

To defeat your enemy you must know your enemy. For individuals who run networks or network services, anyone who attempts to gain or deny access in an illegitimate manner may be considered an enemy. One tool that allows us to learn about this enemy is the honeypot.

A honeypot is a host whose value lies in being compromised by an intruder (see *http://project.honeynet.org/papers* for more details). A "high-interaction honeypot" is nothing more than a regular host that is closely monitored; as an intruder breaks in, the researcher monitors the actions of the intruder on the honeypot.

The key to the honeypot concept is the capturing of intruder activities. For such data to be of use it is critical that an intruder not detect that his or her actions are being captured. If captured correctly this data allows one to identify the tools, techniques, and motives of the intruder.

Today, packet captures using libpcap are the most common data capture technique. Tools that use this technique include Snort, ethereal, p0f, and many more. However, the increased use of session encryption has made packet captures increasingly inadequate for observing attackers. In response to this trend, the Honeynet Project has developed a new tool called Sebek for the circumvention of such encryption. This paper will be an introduction to the Sebek data capture system and the broader impacts of this new type of forensic data.

The Goals of Data Capture

For any data capture technique, we want to determine information such as when an intruder broke in, how they did it, and what they did after gaining access. This information can, potentially, tell us who the intruder is, what their motivations are, and who they may be working with. Specifically there are two very important things we want to recover: the attacker's interactions with the honeypot, such as keystrokes, and any files copied to the honeypot.

Data Capture Techniques and Their Limits

When encryption is not used, it is possible to monitor the keystrokes of an intruder by capturing the network activity off of the wire and then using a tool like ethereal (which is excellent for this work) to reassemble the TCP flow and examine the contents of the session. This technique yields not only what the intruder typed but also what the user saw as output. Stream reassembly techniques provide a nearly ideal method to capture the actions of an intruder when the session is not encrypted. When the session is encrypted, stream reassembly yields the encrypted contents of the session. To be of use these must be decrypted. This route has proven quite difficult for many. Rather than trying to break the encryption of a session, we have looked for a way to circumvent encryption.

Information that is encrypted must at some point be decrypted for it to be of any use. The process of circumvention involves capturing the data post-decryption. The idea is to let the standard mechanisms do the decryption work, and then gain access to this unprotected data.

The first attempts to circumvent such encryption took the form of trojaned binaries. When an intruder broke into a honeypot, he or she would then log into the compromised host using encrypted facilities such as SSH. As they typed on the command line, a trojaned shell binary would record their actions.

To counter the threat posed by trojaned binaries, intruders started to install their own binaries. It became apparent that the most robust capture method involved accessing the data from within the operating system's kernel. When capturing data from within the kernel, the intruder can use any binary they wish, and we are still able to record their actions. Further, because user space and kernel space are divided, there is ample opportunity to improve the subtlety of the technique by hiding our actions from all users, including root.

The first versions of Sebek were designed to collect keystroke data from directly within the kernel. These early versions were the equivalent of a souped-up Adore rootkit that used a trojaned sys_read call to capture keystrokes. This system logged keystrokes to a hidden file and exported them over the network in a manner to make them look like other UDP traffic, such as NetBIOS. This system allowed users to monitor the keystrokes of an intruder, but it was complex, it was easy to detect through the use of a packet sniffers, and it had a limited throughput. The latter made it difficult to record data other than keystrokes.

The next and current iteration of Sebek, version 2, was designed not only to record keystrokes but all sys_read data. By collecting all data, we expanded the monitoring capability to all activity on the honeypot, including keystrokes and secure file transfers. If a file is copied to the honeypot, Sebek will see and record the file, producing an identical copy. If the intruder fires up an IRC or mail client, Sebek will see those messages.

The Sebek Design

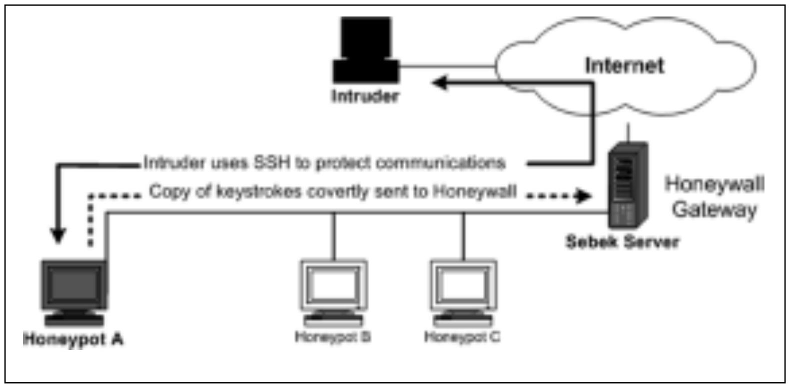

Intruder
Internet
Intruder uses SSH to protect communications
Copy of keystrokes covertly sent to Honeywall
Honeywall Gateway
Sebek Server
Honeypot A
Honeypot B
Honeypot C

Figure 1

Sebek has two components, a client and server. The client captures data off of a honeypot and exports it to the network, where it is collected by the server (see Fig. 1). The server collects the data from one of two possible sources. The first is a live packet capture from the network; the second is a packet capture archive stored as a tcpdump formatted file. Once the data is collected, it is either uploaded into a relational database or the keystroke logs are immediately extracted.

Client Data Capture

Data capture is accomplished with a kernel module, which allows us access to the kernel space of the honeypot. Using this access, we then capture all read() activity. Sebek does this by replacing the stock read() function in the system call table with a new one. The new function simply calls the old function, copies the contents into a packet buffer, adds a header, and sends the packet to the server. The act of replacing the stock function involves changing one function pointer in the system call table.

When a process calls the standard read() function in user space, a system call is made. This call maps to an index offset in the system call table array. Because Sebek modified the function pointer at the read index to point to its own implementation, the execution switches into the kernel context and begins executing the new Sebek read call. At this point Sebek has a complete view of all data accessed with this system call. This same technique could be used for any system call that we may wish to monitor.

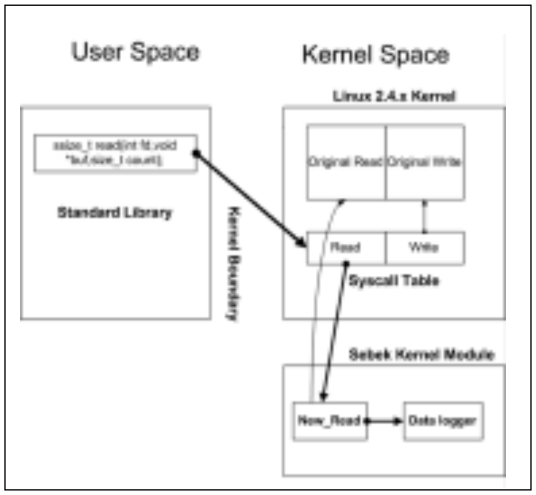


Figure 2

Data that remains encrypted is of little use; to view the data or act on it in some way it must be decrypted. In the case of an SSH session, the keystrokes are decrypted and sent to the shell to have actions performed. This act typically involves a system call. By collecting data in kernel space, we can gain access to the data within the system call, after it has been decrypted but before it has been passed to the process that is about to use it. Thus we circumvent the encryption and capture the keystrokes, file transfers, Burneye passwords, etc.

To make the presence of the Sebek module less obvious, we borrow a few techniques used in modern LKM-based rootkits such as Adore. Because Sebek is entirely resident in kernel space, most of the rootkit techniques no longer apply; however, hiding the existence of the Sebek module is one example of direct technological benefit derived from its rootkit heritage. To hide the Sebek module we install a second module, the cleaner. This module manipulates the linked list of installed modules in such a way that Sebek is removed from the list. This is not a completely robust method of hiding, and techniques for detecting such hidden modules do exist.[1]

1. Phrack issue 61 has an article on detecting hidden kernel modules in its Linenoise section. The article describes a brute-force method for detecting hidden modules by looking for what appears to be the key module structure.

There are two side effects of this removal: Users can no longer see that Sebek is installed and, once it is installed, they are unable to remove the Sebek module without rebooting.

Client Data Export

Once the Sebek client captures the data, it needs to send the data to the server without being detected. If Sebek were simply to send the data to the server over a UDP connection, an intruder could simply check for the presence of such traffic on the LAN to determine whether Sebek was installed. Sebek does send data to the server using UDP, but first it modifies the kernel to prevent users from seeing Sebek packets, not just the packets generated by the local host, but any appropriately configured Sebek packet. Next, when Sebek transmits data onto the network, it ensures that the system cannot block the transmission or even count the packets transmitted.

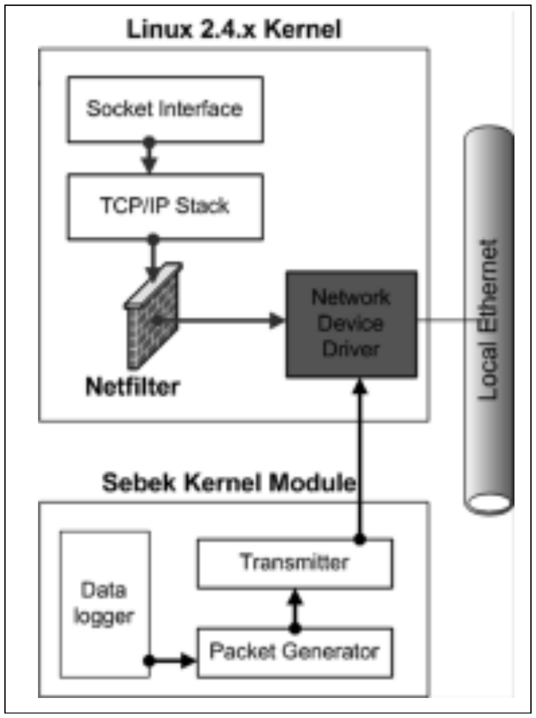


Figure 3

Because Sebek generates its own packets and sends them directly to the device driver, there is no ability for a user to block the packets with iptables or monitor them with a network sniffer. This prevents an intruder on a honeypot from detecting the presence of Sebek by examining the LAN traffic.

The Broader Impact

Not too long after development, it became clear that not only was Sebek allowing us to circumvent encryption, but it was providing a previously unavailable source of data. Sebek was allowing us to look at the honeypot as a glass box rather than a black box. It was easy to monitor the keystrokes of an intruder, but we could also observe the actions of applications that never send data over the network. We initially tried to filter such data, but eventually we realized that such data could help researchers understand the intention and functioning of an unknown and potentially hostile binary installed on a system.

Once these techniques trickle down to the least skilled form of attacker, the script kiddie, it is anticipated that checking for Sebek on a compromised host will be common practice.

We haven't been the only ones to take notice of the potential power of this monitoring technique. Recently, a paper was published on a site purporting to be affiliated with the computer security group Phrack. This paper not only covered the risks associated with running honeynets, but also provided techniques used to detect and disable Sebek. (The honeynet site has a mirrored copy of the article at *http://www.honeynet.org/papers/honeynet/anti-honeypots.txt.*) Once these techniques trickle down to the least skilled form of attacker, the script kiddie, it is anticipated that checking for Sebek on a compromised host will be common practice. The most common technique involves installing a kernel module that attempts to reset the system call table.

The Future

In the near future, the primary goal is to ensure that Sebek is stable and can identify or perhaps even withstand attempts by attackers to disable it. The second priority will be Sebek data analysis. Within the Sebek data we see repeating patterns that are indications of illegitimate privilege escalation. Just as network-based intrusion detection systems examine libpcap data for patterns that represent known bad events, it is anticipated that an IDS based on Sebek data could be developed to detect bad events at the host level. Further, for a few situations it may be that such rules can be contained with the Sebek client, and when the client detects such a situation it would cause the kernel to take remedial action. This would be equivalent to a host-based intrusion prevention system.

Summary

Sebek is a kernel-based data capture tool that was originally designed to covertly monitor activity on a honeypot. Sebek circumvents encryption by capturing the activity in kernel space, where it is in an unencrypted form. Because of this we can capture keystrokes, recover passwords, and monitor any communication including IRC chats, email, and SSH/SCP activity.

Sebek allows an excellent view into the internal activities on a honeypot. Its methodology has not only provided a way to circumvent session encryption but also captures an entirely new type of data. This new data type may lead to the development of new technologies that will help make general-purpose systems more secure.

More information on the Sebek data capture system can be found at *http://www.honeynet.org/tools/sebek/.*

untraceable email cluster bombs

On Agent-Based Distributed Denial of Service

We describe a vulnerability that allows an attacker to perform an email-based denial-of-service attack on selected victims, using only standard scripts and agents. As we will describe, the attack can also target the SMS and telephony infrastructure. What differentiates the attack we describe from other, already known, forms of DDoS attacks is that an attacker does not need to infiltrate the network in *any manner* – as is normally required to launch a DDoS attack. Not only is the attack easy to mount, but it is also almost impossible to trace back to the perpetrator. We describe the attack, some experimental results, and some countermeasures.

by Markus Jakobsson

Dr. Markus Jakobsson is a principal research scientist at RSA Laboratories, and an adjunct associate professor at New York University. He is engaged in research related to wireless security, privacy issues, and protocol vulnerabilities.

mjakobsson@rsasecurity.com

and Filippo Menczer

Dr. Filippo Menczer is an associate professor of informatics and computer science at Indiana University, Bloomington. His research spans Web, text, and data mining, intelligent agents, and complex systems, and is supported by a Career Award from the National Science Foundation.

fil@indiana.edu

The attack involves Web-crawling agents that, posing as the victim, fill in forms on a large set of third-party Web sites (the "launch pads"), causing them to send emails or SMSes to the victim or have phone calls placed. The launch pads do not intend to do any damage – they are merely tools in the hands of the attacker. What makes the attack difficult to prevent is that the launch pads perform the same type of operations as normally desired, thereby giving their administrators no indication that they are part of an attack. This also makes legislation against unwanted emails, SMSes, and phone calls a meaningless deterrent: Without the appropriate technical mechanisms to distinguish valid requests from malicious ones, how could a site be held liable when used as a launch pad? To further aggravate the issues, and given that our attack is a type of DDoS attack, it will not be possible for the victim (or nodes acting on its behalf) to filter out high-volume traffic emanating from a suspect IP address, even if we ignore the practical problems associated with spoofing such addresses.

The attack we describe is an extension of the recent work by Byers, Rubin, and Kormann ("Defending Against an Internet-Based Attack on the Physical World") in which an attack was described where victims are inundated by *physical* mail. While the underlying principles are the same, the ways the attacks are performed and what they achieve are different. Moreover, the defenses proposed in the two papers vary considerably, given both the different threat situations and the different goals in terms of systems to be secured.

Our attack takes advantage of the absence in the current infrastructure of a (non-interactive) technique for verifying that the submitted email address or phone number corresponds to the user who fills in the form. This allows an automated attacker to enter a victim's email address in a tremendous number of forms, causing a *huge* volume of emails to be sent to the victim. (For concreteness, we focus on the email-based attack, noting the similarities to the SMS and phone call attacks.)

Note here that the "double opt-in" defense routinely employed against impersonation of users is not useful to avoid the generation of network traffic. Namely, some sites attempt to establish that a request emanated with a given user by sending the user an email to which he is to respond in order to complete the registration or request. However, as far as our email-based attack is concerned, it makes little difference whether the emails sent to a victim are responses to requests or simply emails demanding an acknowledgment.

It would not be very difficult to mount an attack with, say, a hundred thousand or a million forms.

In a first phase, an agent would harvest forms from the Web by posting appropriate queries directly to some search engine. The agent can then fetch the hit pages to extract forms. For example, at the time of this writing, MSN reports about 5 million hits for the query "free email newsletter" (all terms required) and over 800,000 hits for "send free SMS." However, search engines often do not return more than some maximum number of hits (say, 1,000). One way for the attacker's software to get around this obstacle is to create many query combinations by including positive and/or negative term requests. These combinations can be designed to yield large sets of hits with little overlap.

Once a potential page is identified, it must be parsed by the agent to extract form information. The page value must match a string like "email." Such a heuristic identifies potential launch pad forms with high probability.

In a second phase, forms are filled in and submitted by the agent. This can be performed right after a form is found, or later, using many already harvested forms. Heuristics can be used to assign values to the various input fields. These include the victim's email address and, optionally, other information such as name, phone, etc. Other text fields can be left blank or filled with junk. Fields that require a single value from a set (radio buttons, drop-down menus) can be filled with a random option. Fields that allow multiple values (check boxes, lists) can be filled in with all options. The request can then be sent.

The program could be executed from a public computer, in a library, for example, or a coffee shop. All that is required is an Internet connection. The program could be installed from a floppy disk, downloaded from a Web or FTP server, or even invoked via an applet or a virus.

While it is possible for a site to determine the IP address of a user filling in a form, not all sites may have the apparatus in place to do so. Even if the first phase of the attack takes place from an identifiable computer using a search engine, it is difficult for the search engine to recognize the intent of an attacker from the queries, especially considering the large numbers of queries handled. And it is impossible for a launch pad site to determine how its form was found by the attacker, whether a search engine was used, which one, and in response to what query. In other words, the second phase of the attack cannot be traced to the first (possibly traceable) phase.

We will now report on a number of contained experiments carried out in April 2003 to demonstrate the ease of mounting the attack and its potential damage. We are interested in how many email messages, and how much data, can be targeted to a victim's mailbox as a function of time since the start of an attack. We also want to measure how long it would take to disable a typical email account.

Clearly, these measurements and the time taken to mount an attack depend on the number of forms used. It would not be very difficult to mount an attack with, say, a hundred thousand or a million forms. However, much smaller attacks suffice to disable a typical email account by filling its inbox. Furthermore, experimenting with truly large-scale attacks would present ethical and legal issues that we do not want to raise. Therefore we limit our experiments to very contained attacks, aiming to observe how the potency of an attack scales with its computational and storage resource requirements. We created a number of temporary email accounts and used them as targets of attacks of different sizes. Each attack used a different number of Web forms, sampled randomly from a collection of about 4,000 launch pads, previously collected.

In the collection phase of the attack, we used a "form-sniffing" agent to search the Web for appropriate forms based on hits from a search engine. The MSN search engine was used because it does not disallow crawling agents via the robot-exclusion standard. (We wanted to preserve the ethical behavior of the agent used in our experiments; an actual attacker could use any search engine, since the robot-exclusion standard is not enforceable.) This was done only once.

The collection agent was implemented as a Perl script using no particular optimizations (e.g., no timeouts) and employing off-the-shelf modules for Berkeley database storage, HTML parsing, and the LWP library for HTTP. The agent crawled approximately 110 hit pages per minute, running on a 466MHz PowerMac G4 with a shared 100Mbps Internet connection. This configuration is not unlike what would be available at a copy store. From our sample we measured a harvest rate of 40% (i.e., 40 launch pad forms per 100 search engine hits) with a standard error of 3.5%. At this harvest rate, the agent collected almost 50 launch pad forms per minute, and almost 4,000 forms in less than 1.5 hours. If run in the background (e.g., in the form of a virus), this would produce as many as 72,000 forms in one day, or a million forms in two weeks – probably in significantly less time with some simple optimizations.

The second phase, repeated for attacks of different size, was carried out using the same machinery and similarly implemented code. A "form-filling" agent took a victim's information (email and name) as input, sampled forms from the database, and submitted the filled-in forms. The agent filled in approximately 116 forms per minute. The email traffic generated by our attacks was monitored until the size of the inbox passed a threshold of 2MB. This is a typical quota on free email accounts such as Hotmail and Yahoo. No other mail was sent to the victim accounts, and no mail was deleted during the experiments. When an inbox is full, further email is bounced back to senders and, for all practical purposes, the email account is rendered useless unless the victim makes a significant effort to delete messages. We call kill time the time between the start of an attack and the point when the inbox size reaches 2MB.

In Figure 1 we can observe that for the three smaller attacks (using 514, 1026, 2050 forms), the kill time occurs well after the attack has terminated. For the largest attack (3911 forms), kill time occurs while the attack is still being mounted. This means that the time to disable a standard email account is less than one hour if around 4000 forms are used. If a million forms were filled in parallel by several agents, or by one agent with high bandwidth, we can see from the figure that a standard account can be disabled in well under a minute.

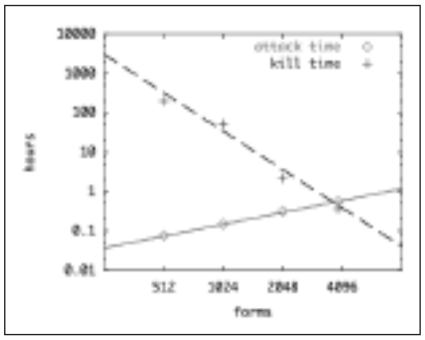


Figure 1

A first line of defense consists of a simple preventive step by which Web sites can avoid being exploited as launch pads in our attack, but without abandoning Web forms. Web sites would use the following simple strategy. After the form has been filled out, the Web site dynamically creates a page containing a mailto link with itself as an addressee. Legitimate users would send the message to validate their request. The email to the Web site would then be used by the site's mailing list manager to verify that the sender matches the email address submitted via the Web form. Although the address of the sender is not reliable, because it can be spoofed in the SMTP protocol, the sender cannot spoof the IP address of its legitimate ISP's SMTP server. The site can thus verify that the email address in the form request matches the originating SMTP server in the validation message.

If legislation is enacted that makes sites liable for any attacks mounted using their facilities, then even poorly behaved sites might wish to employ protective measures to avoid being the defendants in lawsuits by victims of the attack we have described.

There are three caveats to this strategy. First, messages via open relays must be discarded by the site. Second, if an attacker could guess that a user in a given domain requests information from some site, she could request information from the same site for other users in the same domain, potentially spoofing the validation created by the addressee. To prevent such an attack, the validation message created by the site should contain a random number with sufficient entropy that it is hard to guess. Third, one could still attack victims who share their ISP's mail server. In general this would be self-destructive, but a disgruntled employee might use such an attack against his employer. In this case, however, the attack could be traced. With these caveats, our preventive strategy would afford the same security as forms that now request email confirmation, but without sending any email to victims.

The above technique works for forms where a party requests information to be sent to herself, but it does not cover common services such as sending newspaper articles or postcards to others. Sites wishing to allow this can use alternative defenses – namely, well-behaved sites may make the harvesting of forms more difficult by not labeling forms using HTML, but rather, using small images. This would increase the effort of finding and filling in the forms. Given the relative abundance of available forms, potential attackers would then likely turn to sites where no image analysis has to be performed to find and fill in the form. Doing this has no impact on human users, except to a very small extent on the download time of the form.

If legislation is enacted that makes sites liable for any attacks mounted using their facilities, then even poorly behaved sites might wish to employ protective measures to avoid being the defendants in lawsuits by victims of the attack we have described.

In our full paper, available from *http://www.markus-jakobsson.com*, we describe further experimental results, along with descriptions of secondary lines of defense that can be used by end users to avoid becoming the victims in attacks where potential launch pads have not taken sufficient precautions to prevent a successful attack from being mounted.

insider threat

Models and Solutions

by Mudge

Mudge continues in his goal to "make a dent in the universe."

mudge@intrusic.com, mudge@uidzero.org

When a problem persists even after the outpouring of tremendous sums of money and resources, it is sometimes necessary to revisit the belief systems around what your problem might actually be. Intrusion detection systems, security scanners, managed firewalls, and external audits have all provided some form of value, but have they addressed the issues they were deployed to solve? In cases where they have, has it been to the extent hoped for and expected?

Several very large organizations have approached me with this dilemma recently. They are finding themselves overrun with reverse tunnels. In actuality, it is not the reverse tunnels that are the problems as much as the compromised internal systems. Identifying reverse tunnels, and various covert communications channels, can be difficult in certain cases. However, the majority of instances are very easy to identify accurately.

It is the purpose of this article to share a perspective on security within an organization's perimeter, using a perspective and threat model largely derived from counter-intelligence/counter-espionage (CI/CE) models. The various solutions, such as some of the reverse-tunnel analysis below, are derived from a framework I have constructed called "The Physics of (Internal) Networks." Together they accurately define and map the networked "insider threat" issue. It is important to point out that this paper only targets *internal* corporate networks.

Before we embark upon the description of reverse tunnels, HTTP in particular, and some methods to identify these within your network, let us look at some of the current industry beliefs.

Increasingly, the industry believes the threats to protect against are the overt attacks that might be launched against them in the future. The attacks being worried about are directed specifically against them. Further, it is believed that the attacks will originate externally and will attempt to breach the firewall perimeter. What the attacks will attempt to accomplish does not seem to be an area that has been given much thought, the predominant belief, stemming from popular media reports, being that of disrupted service or various kinds of Web defacement.

Perhaps, whether accurate or not, it is too painful for organizations to entertain the notion that they might already be compromised. Being overrun by reverse HTTP tunnels might be an easier pill to swallow than accepting that these reverse tunnels are symptoms of actions initiated from internal machines that are already compromised.

Attacks draw unwanted attention. It is, and always has been, preferable in most situations to use credentials that are permitted on a system, however those credentials are obtained. This way, there is no actual "attack" as IDS would classify it.

Like a mole in a government agency, the greatest value is achieved through unnoticed longevity in the target environment. The expected movement and characteristics of information and its handling related to business functions must change in these cases, providing us with the ability to identify such covert activities. Profiling the business functions and their information flows on the internal network is the important component, not profiling the people.

How Much Progress Have We Really Made?

What follows is a subset of various trojan and back-door tools and targets, along with some time frames showing when the author of this article first came across them. The items mentioned below have been found in use, unmodified or trivially altered, up to the present – very successfully. Intentionally, only tools that have been around for many years are listed. The greater concern does not reside in the actual modified programs and tools themselves but, rather, in the fact that they are still so tremendously successful, and seldom spotted until after it's too late.

NAME	BRIEF DESCRIPTION	ROUGH DATE
fingerd	accepts commands to add users, launch an interactive root shell, etc.	1994
BSD-logind	embedded password that allows and hides a root-level login	pre-1997
rshd	back-door account with root access	pre-1997
Telnet	trojan to copy username, hostname, password of anyone connecting to a remote machine	1993
Telnetd	back-door enabled through Telnet option negotiation variables (placed into various distribution trees)	pre-1997
ICMP (pinsh/ponsh)	covert communications over ICMP echo packets	1995
Ident	back door	1995
dynamic library trojan/(kernel interface calls)	hides processes the interloper has tagged (would and still does defeat many host-based intrusion systems)	1993

The success and longevity these sorts of tools enjoy highlights the fact that the internal network threat model is not being addressed by current network intrusion detection solutions.

Are We Under the Belief That the Sun Orbits the Earth?

Consider the following data points that go hand in hand with the tools and techniques just mentioned:

- Intruders are already inside most corporations, often sitting on key components of critical infrastructure and usually without knowledge of exactly what they are in control of.
    - Accidental catastrophic failure is possible.
    - Intentional catastrophic failure is possible.
- Passive control of systems is much more desirable than disruption or damage without purpose.

- Target selection is opportunistic.
  The selection is often acquired from within a large selection of systems, usernames, and passwords of already compromised systems:
    - VPN – scanning DSL/cable/dialup (also known as Island Hopping)
    - Sniffed credentials of corporate accounts accessed from schools/universities (Fluffy Bunny demonstrated and documented this in his compromise of Akamai and other substantial environments)
    - Shell systems or other large user-base machines through trojaned binaries/applications
    - Sniffed credentials obtained via compromised systems at ISPs

- Passive control and tools have not changed much since pre-1996.
- Cloaking tools have not changed much since pre-1996.

These last two points are not news to the people involved in operational security and cleanup. With all of the updates and advances that the defensive products being deployed incorporate, the same rootkits and hide packs are consistently found to be running on compromised systems.

Obviously, the issues at hand goes well beyond simply identifying tunnels and reverse tunnels. However, it remains important to address and be able to identify symptoms of such problems. Here are a few ways to analyze internal network traffic to identify streams as likely being reverse HTTP tunnels. Again, these are just a few ways to look at network traffic dumps that have proven successful for this purpose.

A Quick Definition of Tunneling

Tunneling is the process by which one communication channel is embedded within another. Tunneling is often performed not only to hide a session's contents from casual observation, but to allow compromised hosts located on an internal network to use firewall- and filter-allowed protocols in order to act in collusion with outside agents. HTTP tunneling encapsulates data in HTTP; often the data is simply sent across the ports associated with HTTP and not even embedded within the protocol itself.

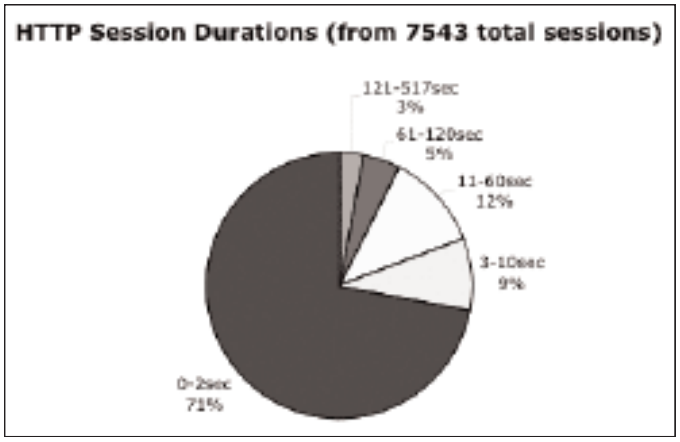
Freely available software to help automate the planting of back doors is in wide circulation. Once compromised, the internal systems are able to communicate with external targets while appearing to be standard Web surfing or other allowed activities. The more common modus operandi utilizes a variant of HTTP tunneling known as reverse tunneling. In this case, what appears to be a client system surfing the Web contacts a specified Web server and allows commands to be sent back to it. Thus, the client becomes a server to the intruder's external system.

The key to discovery lies within the understanding of how things work normally (remembering that this paper is specifically dealing with internal networks, CI/CE, and the insider threat). Reverse tunnels' primary purposes are to permit a single actor to:

1. enable their communications to pass through outbound filters,
2. camouflage the connection, and
3. allow control or influence to originate from external locations.

The above do not adhere to the "Physics of (Internal) Networks" as defined by business function or data purpose. So, taking the tcpdump or other sniffer logs from your internal networks, we can begin. (You do keep these sorts of things handy or at least have such network traffic logging systems deployed, don't you?)

Quick, Dirty, and Successful Reverse Tunnel Analysis Techniques



Duration

HTTP sessions are usually short-lived and initiated per-page (or per-item). A session to port 80 that lasts more than a few minutes is quite unusual for standard Web surfing. However, a session of this duration or longer is quite common for interactive shell connections.

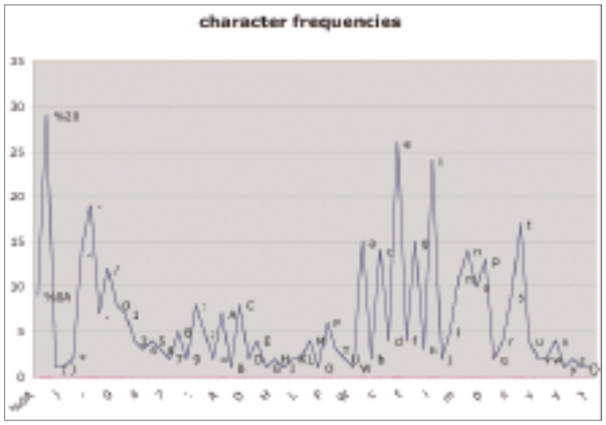
Client-Server Flow Directions

HTTP operates in a client-server fashion. The browser acts as the client and typically consumes more data than it produces. Client systems that produce significantly more data than they consume in a session can indicate potential reverse-tunnels.

Lack of Client Browser Identification to the Web Server

When a client connects to a Web server, the browser sends not only the request for the Web page but a series of directives. The following is what the OmniWeb browser on a MacOS X system sends.

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.75C-CCK-MCD {C-UDP; EBM-APPLE} (Macintosh; I;
PPC) OmniWeb/v496
Host: 127.0.0.1:8080
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
        image/png, image/tiff, multipart/x-mixed-replace, /;q=0.1
Accept-Encoding: gzip, identity
Accept-Charset: iso-8859-1, utf-8, iso-10646-ucs-2, macintosh,
        windows-1252, *
Accept-Language: en, *;q=0.5
```



If the first data packets in the session sent from the client could not possibly represent something similar to the character-frequency graph above, the session is potentially suspect. Bi-grams, tri-grams, and character frequency are all well-understood cryptography and linguistics analysis techniques that work very well here.

Many permutations on the graph exist. Was too little data sent from the client initially to form a normal request? Did the client never attempt to send this sort of initial data (i.e., server sends first payload)? And so on.

Interactive Versus Non-Interactive Data Streams

Surfing the Web seems to the end user to be an interactive experience. The user requests a Web page, is presented with information, and, based upon the options within this new information, performs subsequent requests or actions.

The system-to-system communications which make up each stream are in fact non-interactive in comparison to Telnet and others.

Reverse HTTP tunnels are most frequently interactive sessions allowing "server" terminal or shell-style communications with the initiating "client."

This is easily spotted by, among other things:

- Small data packets making up most of the "server's" data
- Large deviations/variances in the time span between packets
- Both large and small data packets making up the "client's" data stream where there are distinct groupings of large vs. small

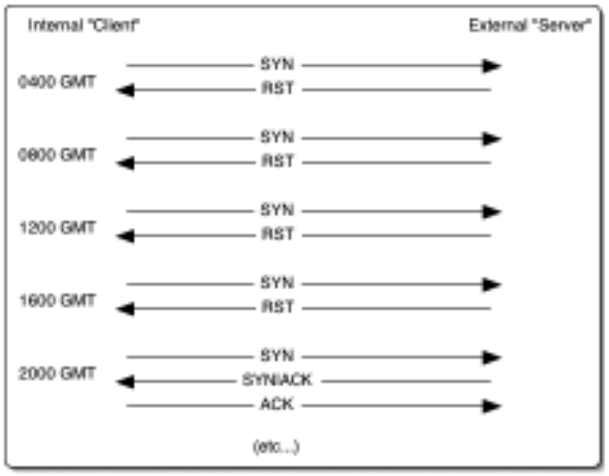
The reader is referred to Yin Zhang and Vern Paxson's paper[1] on this topic.

1. Y. Zhang and V. Paxson, "Detecting Stepping Stones," *Proceedings of the 9th USENIX Security Symposium*, (USENIX Association, 2000) *http://www.usenix.org/events/sec2000/zhangstepping.html.*

PERIODIC REQUEST SPACINGS

Cron or other timed automated execution methods are commonly used on the compromised internal system. The internal system in these situations attempts to connect to the external system once an hour, once every several hours, once a day, etc. When the external system that is acting in collusion accepts the connection, the client presents the equivalent of a shell prompt. Connection attempts to systems that are rejected most of the time but are successful on occasion is another potential indicator of a compromised system.

The figure on the right shows connection requests at four-hour intervals, each being reset by the server system. The final connection proceeds as one would expect. A common permutation of this leaves the initial SYN packets unanswered.



Wrap-up

While this subset of methods is useful in spotting reverse HTTP tunnels, individually they still offer a potential for false positives. Luckily, more than one of these individual checks will almost always have to be true in the actual tunnel situation. Combining these checks and others in logical ways can easily negate most occurrences of false positives. A commercial tool to address these and other insider threats will be available at *http://www.intrusic.com.*

life without firewalls

by Abe Singer

Abe Singer is a computer security manager at the San Diego Supercomputer Center, and occasional consultant and expert witness. He is responsible for operational security for SDSC, involved with the Teragrid security working group, and is doing research in security measurement and logfile analysis"

Abe@SDSC.edu

Here at the San Diego Supercomputer Center (SDSC) we recently went almost four years without an intrusion on our managed machines.[1] And it was some time between that one and the previous one.

We have a decent-sized organization – a few thousand machines, over 6000 users worldwide. We think we do a pretty good job of keeping our systems secure.

And here's a little secret: We don't use firewalls.

Why not? Well, because, basically, they're not worth it, and they don't provide us with the protection we need. Because it's about *effective* security – solutions which actually work against actual threats, which scale, and which are robust.

This article outlines our strategy for effective security. While not all of it is applicable in all environments, I hope some (maybe a lot) of it is useful to others.

Space limitations prevent me from going into detail about our implementation. Expect more in future articles.

1. We do have "unmanaged" (user-managed) machines that have been compromised. This will be explained in more detail below.

The Myth of Firewalls

There is a pervasive myth that firewalls are necessary for effective security. Firewalls have become a panacea, and are assumed to magically protect everything. The net result is often that a network is *less* secure.

"Use a firewall" is a common recommendation from security literature and practitioners. VISA's Cardholder Information Security Program requires online merchants to "install and maintain a firewall."[2]

2. VISA Cardholder Information Security Program, *http://usa.visa.com/business/merchants/cisp_index.html*.

The myth that a firewall is necessary for effective network security is so prevalent that many believe you are doing something wrong if you don't have one.

For example, I was helping a private research lab construct a comprehensive security plan, which focused on infrastructure protection. They hired a new CTO, who informed us that he wanted a firewall, because whenever he discussed security with his peers at other organizations, they were incredulous that he did not have one, and he felt that his reputation was suffering as a result. Some time later, at a technical staff meeting, the issue of security came up, and he announced that they were now secure because they had a firewall. (They weren't, and some of the staff called him on it.)

I used to be a full-time consultant, and I can't tell you how many times a customer (or potential customer) would say something similar: "We're safe – we have a firewall."

But firewalls *aren't* the magical bullet-proof vest that the public seems to think they are. Just look at the spread of worms throughout the Net: Code Red, Blaster, etc. Look at Web sites that have been defaced by vandals, or had credit card or social security numbers stolen. Many of these compromised sites had firewalls.

Why does the myth persist? First, firewalls are sexy. It's much c00l3r to play with a firewall than to patch machines. Second, we live in the "magic pill" culture, where we look for a one-time quick fix, and management wants to hear that the problem is solved.

Why Firewalls Don't Work

So what is a firewall? Early firewalls were just packet-filtering routers, which could only filter IP addresses (not port numbers). Then came port filtering, proxy firewalls, and stateful packet filtering (SPF). To some people, using NAT is considered a firewall.

A firewall works only as well as it's configured. A firewall which allows traffic through on every port provides no protection whatsoever. One which does not allow any traffic through will probably protect against external attacks, but isn't otherwise useful.

A firewall works only as well as it's configured.

So a firewall has to allow some traffic through. And if the machine receiving the traffic behind the firewall has a vulnerability, it can be compromised.

Now at this point the firewall-o-philes will be saying, "Oh, but a proxy firewall would solve that problem" or "Stateful packet filtering is the answer." Those technologies also only work as well as they're configured or programmed.

A proxy has the *capability* to examine the data and only allow "legitimate" data through, but it has to be programmed to differentiate between what is legitimate and what is not. And that program has to be maintained, bugs have to be fixed, and so on.

And you still have to protect the service against attacks from other hosts on the same network *behind* the firewall.

Firewalls which rely on chained rule sets are vulnerable to cascade failures – a change in one rule can have an effect on every rule which follows it. I've seen systems that relied on a firewall to block services which were only supposed to be available on the local network, but which were made available to the entire Internet due to the unforeseen result of a firewall rule change.

Firewalls also have performance issues; they usually only work if all traffic is directed through them, and usually not at the speeds found, for example, on OC-192 connections. Firewalls do not work well or at all in a high-speed network with multiple paths.

Finally, having a firewall should not be used as an excuse for poor system administration.

Now, having said all that, I'm going to back down a little and say that firewalls do have their uses. They're just not the magic pill that people think.

Lack of Metrics

The truth is we don't have any real scientific measurement of how effective different security practices are. There's no data that shows that, to make up an example, firewalled sites have 34.5% fewer compromises than those without. There's no data that shows how much longer it will take, on average, to compromise a "secure" system configuration vs. a default "out of the box" configuration.[3]

3. Marcus Ranum's cat may have some numbers.

(We could really settle some religious wars if we *did* have such data.)

SDSC's Approach

History

Like almost everyone, we had to feel the pain before getting smart about security. About 10 years ago, SDSC had recurring problems with intruders. As fast as an intruder could be kicked out, another one got in. Finally, we decided to take drastic measures: We shut down the entire network. Machines were reinstalled and not allowed back on the network until they were considered secure. Our Cray was down for three weeks, the longest it had ever been down (before *or* after). Our director, Sid Karin, said, "I never want to do this again." Meaning that whatever we did should work over the long term, not just be an immediate fix.

A security policy and strategy should be based on a realistic risk analysis.

Sid also said that we should have an open environment, where people can do what they want to do. (He actually said that if someone wants root access, why not give it to them?) But he recognized that one person's actions on one machine can have an impact on other machines on that network.

So we developed a long-term strategy to keep our machines and network secure, while providing resources that are open and usable. We did not do everything immediately. Rather, we implemented what we could, and improved things as new technologies became available. We also recognized that there are people who just want to do their own thing, either because they have a need that doesn't fit with our environment or because they have an ego problem, so we also provided a way for those people to manage their own resources.

The SDSC Environment

SDSC is a facility which provides supercomputing resources for scientific research, and does research in high-performance computing technology.[4] SDSC is also part of TeraGrid and Internet2. SDSC does not do any government-classified work.

4. The San Diego Supercomputer Center, *http://www.sdsc.edu*.

We have about 6000 users, of which only about 300 are on-site. The rest are at other institutions around the globe (mostly in the US).

We have several thousand systems on-site and about five petabytes of near-line storage, plus several hundred terabytes of spinning disk on a SAN. Our network supports 10Gb Ethernet (no, that's not a typo) internally and to other TeraGrid sites. We have multiple OC-192 connections to various sites. In other words, LOTS of bandwidth.

We do not insist on absolute homogeneity (more on that below). On-site users get their choice of desktop: Linux, Solaris, Windows, or Macintosh. Many users have more than one desktop machine. Our infrastructure machines are a combination of Linux and Solaris. Oh, and we have some IRIX machines used by the Visualization group.

We currently support the following OS revs:

RedHat 7.2, 7.3, 8, 9; Solaris 7, 8, 9; IRIX 6.5; Windows NT4, 2000, 2003, XP (for some specific applications); MacOS 9, X

We treat hardware as commodity devices. System configuration is independent of the hardware. If a system dies, it is replaced with new hardware and auto-installs. User downtime is kept to a minimum, as are support staff resources.

Risk Analysis

A security policy and strategy should be based on a realistic risk analysis. Our analysis looks at the assets we are trying to protect and the threats to those assets.

ASSETS

To SDSC, the most important thing to protect is the integrity (and, where necessary, the confidentiality) of our data. In this context, "data" means both user and system data (mess with system data and you've compromised the machine).

We also need to protect our resources: bandwidth, CPU, and data storage capacity. Many of the script kiddies out there don't want our data (or yours), they just want our disk space and bandwidth.

Finally, we want to protect our reputation as a site that does security well and provides high performance and reliability to its users.

THREATS

We have several threat vectors to deal with. First, there are the Evil Internet Hackers™, who want to break into our machines for whatever reason. As mentioned above, some just want to use our resources, but some are targeting us specifically because we've got a high profile,[5] such as those that hold us responsible for Kevin Mitnick's most recent incarceration (as of this writing ;-).

5. Probes of takedown.com, *http://security.sdsc.edu/incidents.*

In addition to external, anonymous threats, we have to worry about our own users, whom we really don't trust any more than the "outsiders." Sometimes our users willfully violate policy because they just don't care. Others are light on clue.

And sysadmins can do stupid, careless, or malicious things; we have to find a way to contain them, too (I think I hear the sound of cats and a herder).

POLICY

The general SDSC security policy is pretty short. It basically says that the security policy is to protect the confidentiality and integrity of our users' data and to provide reliable service.

In order to provide this protection, we must protect our file servers. Our policy requires that only machines considered "secure" according to a reference system can be on networks that can communicate with the file server. We refer to these systems as "managed" machines or "reference systems." How we build a reference system is described below.

We also realize that accounts can be compromised by the sniffing of passwords, and that an account compromise can lead to a system compromise. So we instituted a policy that says that no authentication protocols which use plaintext passwords or other secrets which can be intercepted and reused can pass between our "trusted" networks and other networks. I call this the "no plaintext passwords" policy.

Since a compromise of one system can lead to compromise of other systems, our policy also requires anyone with a system on any of our networks to report any suspected compromise to the security group for investigation.

For privileged access, the user must explain why they need the access and sign an acceptable-use agreement that is also signed by their supervisor. The root password is only given out where absolutely necessary, such as to people who need to log in to a system console during the installation process. For all other cases, limited "sudo" access is given for UNIX access, and a local administrator (not domain administrator) is provided for Windows users. The user is also given a little lecture to make clear that they are not to use their privileges for activities beyond the reason they gave for access – that access/ability is not authorization. The agreement form also indicates this.

Finally, in order to accommodate users who do not want to comply with our reference guidelines, we have a section of the network which we call "The Outback," in which anyone can install a system. The user and their supervisor must sign a form indicating that they take responsibility for their system, and that if it is compromised we may take custody of the system for forensic analysis and, potentially, as evidence in a criminal case.

We have, on occasion, had to strip a user of privileged access. We also have seized numerous Outback machines over the years (although the rate *has* decreased over the last year or so).

SDSC's Security Approach

Our security approach revolves around a few basic strategies described here.

Scalable Configuration Management

Most (all?) default vendor installations of operating systems have security vulnerabilities. Systems with default installations will eventually get compromised.[6]

6. See *http://worm.sdsc.edu/*.

Another common point of security failure is secure configurations that are lost after a system is reinstalled, upgraded, or overwritten by a vendor patch. Also, sometimes the configuration change fails to get installed on all machines – 99 were patched, but the 100th forgotten, or a machine that was down for several weeks and stored in a closet is booted onto the network, and it does not have the fixes that were put on the other running systems.

Our system configurations are based on a "reference system" and managed using automated configuration management software (cfengine).[7]

7. cfengine, *http://www.infrastructures.org/cfengine/*.

We use cfengine to correct things that the vendor gets wrong, to install locally built software, and to install/maintain configuration files (e.g., fstab). Every system runs cfengine upon boot, and each night, files or permissions that have been modified by hand on a system will be detected by cfengine and restored to their "reference" state – the system is effectively self-healing. Cfengine operates on "classes" of machines – a change put into the reference will be automatically installed on all machines in that class. And since cfengine logs the changes it makes and makes backup copies of any files it modifies, it serves as an automated intrusion detection and recovery system.

Now, those of you who paid attention when I was ranting about firewalls might point out that a misconfiguration will cause a failure across *all* systems; that is correct. But since most host-based protective measures are not interdependent (chained), we are not as susceptible to the cascade failures mentioned above. And, yes, we could open up a window of vulnerability on our machines. But we think the trade-off is worth the risk, as we don't end up with machines on the network whose configuration state is unknown. And putting the fix into the reference guarantees that it will be propagated to all machines.

Aggressive Patching

Most publicized compromises (especially worms) have taken advantage of vulnerabilities for which patches were already available, in some cases for months or years. Aggressive patching could probably solve 90% of most companies' security problems.[8]

8. Marcus Ranum's horse would probably disagree with that number.

Security patches are installed as soon as possible. We prioritize patches based on a combination of whether the vulnerability is remote or local, and whether or not an exploit exists.

Patches are tested on a single machine, then on willing victims' (users) desktops for Microsoft systems, and then are distributed to all appropriate hosts with automated tools.

NO PLAINTEXT PASSWORDS/STRONG AUTHENTICATION

The other vector for attack is via a compromised account where the password for that account had been sniffed from the network, or guessed. Additionally, many intruders will install a sniffer, regardless of their main purpose, to opportunistically find other accounts and machines to compromise.

A compromised account is one of the most difficult to detect. How can one easily determine whether a given login session is the legitimate user or an intruder?

We use a combination of solutions to provide authenticated services where passwords are either encrypted or not transmitted (e.g., SSH, Kerberos).[9]

9. See Abe Singer, "No Plaintext Passwords," *;login:*, November 2001, vol. 26, no. 7, for details of how we eliminated plaintext passwords. *http://www.usenix.org/publications/login/2001-11/*.

Practices

REFERENCE SYSTEMS

The general process of creating a reference system is to first install an appropriate selection of system software (usually, the vendor's procedures will be used for this), then add SDSC-specific software (e.g., cfengine), then modify everything to fix security problems and establish the functionality we require. Key to making this work is a high degree of automation, which allows the easy replication of the system.[10]

10. Jeff Makey wrote this in an unpublished document describing our reference system.

Our reference system includes the following:

- Automated configuration management using cfengine (see description, below).
- Time synchronized to atomic clocks using Network Time Protocol. This is essential when mounting a central NFS file system. Synchronized time is also useful for forensic purposes in analyzing file timestamps and correlating syslog entries from multiple hosts.
- Centralized account management. NFS requires that UIDs be consistent across clients in order to prevent inadvertent access to protected files. Centralized account management keeps UIDs consistent across all our machines.
- A password-changing program that rejects easy-to-guess passwords. Our password changing uses cracklib[11] to test passwords that could be guessed with crack, instead of trying to crack them after the fact. Why look for crackable passwords when you can prevent people from using them in the first place?
- Detailed logging to a central host. All syslog facilities are forwarded to a central loghost and archived on our storage system (yes, we have eight years of logs stored!). Centralized logging preserves log data in the event that a system is compromised and local copies of the logs are modified by the intruder. Centralized logging also allows us to monitor logs for interesting activity across all hosts.
- Most services (including RPC) protected by TCP Wrappers. Some services which should only be used within our networks are limited to just those networks. TCP Wrappers also provides consistent logging of accepted and refused connections for services, even those accessible from anywhere.
- SSH.[12]
- Kerberos 5 authentication for Telnet, FTP, rlogin, & SSH.[13]
- Email notices sent to administrators at shutdown and boot time. Getting boot and shutdown notices helps alert us to potential problems with a host. It also reminds us to check hosts that have been down for a period of time and ensure that they are fully patched.
- Sudo.[14] Most users who need privileged access are given it via sudo. Very few people actually have the root passwords. Sudo assists with accountability by logging

11. Cracklib, *http://www.crypticide.org/users/alecm*.

12. See Singer, "No Plaintext Passwords."

13. See Singer, "No Plaintext Passwords."

14. Sudo, *http://www.courtesan.com/sudo*.

15. Sudo can be subverted. We handle this through policy.

the commands performed as root, plus it allows us to limit what the user can do, where appropriate.[15] Additionally, keeping the list of people with the root password to a minimum keeps us from having to change the root password quite so frequently.

- Identd with encrypted responses. This tool allows a remote site to collect ident information, but that information is encrypted and does not provide useful information to an attacker. However, if the remote site has a problem with connections from *our* site, they can send us the encrypted ident string which *we* can decrypt and use to track down what is happening.
- A modified "xhost" program with "+" functions disabled. The "+" function authorizes *any* connection from a remote host, regardless of who owns the process at the other end.
- A version of "su" that uses group 0 as an access list. Some vendor versions of "su" only allow users who are in group 0 (called "system," "root," or "wheel" on some OSes) to su to root. Those that don't have the vendor version replace with one that does. This helps prevent an unauthorized user from becoming root even if they *have* the root password.

Building a Reference System

We have a reference for each operating system we manage. We start by installing the OS, using the vendor's installer. We then remove software that we don't need and that (1) starts daemons, (2) has setuid/setgid programs, (3) has files with ownership or permissions we don't like, or (4) is duplicated by our own versions. We reboot the system to make sure everything comes up as we expect, and lather, rinse, repeat, until the system is configured the way we like.

We then build an auto-installer using the vendor's software (e.g., kickstart for RedHat) for the system as configured. The auto-installer also installs all necessary patches and runs cfengine once the base installation is done. Cfengine is put in a startup script to run at boot time and as a cron job to run once a day.

Centralized Data Storage

Key to keeping secure reference systems is maintaining the integrity of the reference. Additionally, maintaining the integrity of user data is necessary.

We use a central NFS file server. Since the NFS protocol has some weaknesses (file handles can be sniffed/guessed), we only let the NFS server talk to hosts that are considered "secure" – reference systems. File systems are exported to the client's IP address in order to avoid DNS spoofing.

Furthermore, the NFS server does not have a default route – it only has routes to the "trusted" networks which only have hosts that we manage. An attacker is not able to establish a two-way connection to our NFS server.

The reference partition of the file server is export read-only to all hosts, so that a compromise of a host cannot be used to compromise the reference. Administrators must be able to log in to the file server in order to make changes to the reference.

We export file systems read-only where possible, but other file systems on the file server, such as user home directories, are exported read-write. The root user on all clients is mapped to "nobody" on the file server, since root on one machine could be used to compromise files exported to another machine.

Windows Systems

I'm not going to say a lot about Windows here, only a little bit about how we manage them . . .

We use Ghost and SMS to install and manage our Windows system: Ghost to install systems, SMS to install patches on existing systems.

I don't think firewalls are completely useless.

A Place for Firewalls

Okay, as I said above, I don't think firewalls are completely useless. It's really about proportion of effort. I think most organizations spend more than 90% of their effort (money *and* time) on firewalls, and it should probably be less than 5%. Firewalls can provide an extra layer of protection (provided you know what you are protecting *against*). And some people say that firewalls are for machines that cannot protect themselves, such as printers and maybe Windows machines.

We *do* perform some packet filtering on our network. We have anti-spoof filters (on ingress) to prevent someone on the outside from sending packets that appear to come from the inside. We keep Windows machines on their own subnet and only allow certain Microsoft protocols within that subnet.

We are also playing with firewalls for some applications. One is for users who bring in their laptops. We currently put them on an open, external network like The Outback (see above). We are experimenting with a firewall that allows outbound connections but no inbound, to provide some measure of protection for the user machines. The firewall may reduce their exposure, and it provides us with a choke point to monitor and block misbehaving machines from attacking the rest of our network.

We may also use a firewall for the Windows network, as new attacks pop up so frequently, and some of them are difficult or impossible to control from the host.

Conclusion

Firewalls don't necessarily provide as much security as popularly believed. Securing individual hosts can provide better security and functionality than using a firewall. Hosts are protected from each other in addition to the Internet. Use of scalable configuration management, no plaintext passwords, and aggressive patching can provide host-based security in a scalable, cost-effective manner. It has worked for us; maybe it can work for you.

nessus

by Renaud Deraison

Renaud Deraison is the original author of Nessus and the manager of the Nessus project. Nessus is the world's most popular vulnerability scanner and has more than 50,000 users.

deraison@nessus.org

and Ron Gula

Ron Gula is the CTO and co-founder, with Renaud Deraison, of Tenable Network Security, which funds the Nessus project and produces enterprise security management tools, as well as a variety of commercial active and passive vulnerability scanners.

rgula@tenablesecurity.com

The basic concept of conducting a network vulnerability assessment is to find security flaws so that they can be fixed before they are exploited. Nessus is a free, powerful, and easy-to-use network vulnerability scanner. It is an open source tool that is available at *http://www.nessus.org* for most UNIX operating systems.

Like most vulnerability assessment tools, Nessus performs a network vulnerability audit by first determining which hosts are alive, which applications are present, and then which vulnerabilities may be present. It has many techniques to determine all of this information and is constantly updated with new features and new vulnerability checks. Nessus also takes a closer look into the network and can identify unneeded servers or services that can have a greater long-term impact on security.

Nessus Architecture

Nessus can be deployed in a permanent or stand-alone configuration. Nessus installations consist of one or more "Nessus daemons," which are servers that perform the actual vulnerability testing, and one or more "Nessus clients," which are the control points to launch scans and review the collected data.

For single-host Nessus installations, such as a laptop, a user would install both the Nessus client and the Nessus daemon. In order to run a vulnerability scan, the Nessus daemon needs to be started first and then the Nessus client can be launched.

The client-server architecture of Nessus also makes it possible to deploy permanent "scanners" and have multiple users share them as a resource. For example, a Nessus daemon could be installed on a secured OpenBSD server located outside of a university network. This Nessus daemon can be connected to any Nessus client (with proper credentials such as a username/password or a certificate) to perform a scan. Multiple users can perform these scans simultaneously. Each user account on a Nessus daemon can also be restricted such that they can only scan a specific range of IP addresses.

Nessus Attack Scripting Language

The heart of Nessus's power and flexibility is the Nessus Attack Scripting Language (NASL). This interpreter is yet another computer language, but this one has the advantage of being a portable language designed to do network vulnerability checks. The syntax of NASL is beyond the scope of this article, but full documentation and almost 2000 examples are available at the Nessus Web site.

In version 2 of Nessus, this language was completely rewritten to improve speed and performance. A variety of built-in functions were also added, such as the ability to evenly slice IP ranges rather than just scanning sequentially. Almost any type of network vulnerability action can be coded with NASL. Some examples include generating a specific packet for a DoS attack, completing a TCP handshake to observe a banner string, and completing an NTLM authentication to a Windows primary domain controller.

Individual NASL scripts can be run alone. This is useful for testing, or building a set of scripts which are not run by the Nessus daemon. To test a NASL script, simply run the command `nasl –t <IP Address> <script name>` and the script will be attempted. What is more useful is that the entire processing of NASL can be viewed with the `-T` option, which can trace each step of the script and record it to a specific file for analysis. This makes writing custom NASL scripts and modifying existing ones extremely easy.

All Nessus plugins are open source and can be inspected by the Nessus user community. The Nessus project also attempts to maintain each original author's copyright information.

Installing Nessus

Nessus source code, pre-built binaries, and even automated Internet installations are available from a variety of mirror sites and from *http://www.nessus.org*. All Nessus build scripts are very robust and will complain if certain packages are missing from the desired target platform for Nessus.

On most UNIX systems, the easiest (and most insecure) way to install Nessus is with the simple command

```
lynx –source http://install.nessus.org | sh
```

Though easy, this command allows an attacker who has gained control or influence over your DNS service to redirect you to another site and provide different code for you to execute.

A more secure way to retrieve Nessus is to download the nessus-installer.sh tool from the Nessus Web site. This shell archive is a complete Nessus source distribution, which also automates the build process. PGP-signed MD5 signatures are also provided at the site for verification of the distributed content.

A compile-time decision that needs to be made is whether the X Windows Nessus client is needed. Without X Windows, there is no GUI for the Nessus client. This is not a problem, since Nessus is readily configurable from the command line and also from a variety of Nessus clients (some for Microsoft Windows platforms) that are maintained outside of the Nessus project. If an X Windows GUI is desired, then the graphical toolkit (GTK) library needs to be present on your desired flavor of UNIX. Most fully loaded installations of Linux or RedHat have the required libraries already installed. There are additional directions to install and compile the Nessus client with GTK support at *http://www.nessus.org/nessus_2_0.html*.

Once Nessus is downloaded and compiled, it will prompt the user for a few post-installation configuration options. These will prompt the user to generate a unique cryptographic certificate for the Nessus daemon and to create at least one Nessus "user" for the Nessus daemon. This user account will be used by the Nessus client to log into the Nessus daemon and launch a scan.

Starting Nessus and Launching a Command Line Scan

To start the Nessus daemon, simply use the command

```
nessusd –D
```

This will invoke the Nessus daemon and leave one process running. If the Nessus daemon is performing a scan, it will fork and several dozen Nessus daemon processes may become active.

By default, the Nessus daemon will listen for Nessus client TCP connections on port 1241. Nessus's man pages also do an excellent job of describing the location of the various configuration and logfiles for the Nessus daemon.

When Nessus is installed, it comes with the most recent vulnerability checks available. In the future, new vulnerability checks will be downloadable from the Nessus project by typing the command nessus-update-plugins. If you are behind a Web proxy, this script needs to have some internal variables edited such that it can access the Internet through your proxy. Some Nessus users choose to create a cron job to schedule a download of the updated Nessus plug-ins once per day or once per week.

The Nessus command line client has a variety of features, and there are many ways to launch a scan. What follows is a very quick example to illustrate how easy it is for Nessus to do a scan.

To verify that the Nessus client and Nessus daemon can talk, we will invoke the client to ask the daemon for a list of scanning sessions (of which there should not be any). If there is a communications problem, such as the wrong port, IP address, username or password, we will get an error. If things are configured correctly, we will then move on. Here is the command to try to log onto our Nessus daemon (on the same server) with an example username and password of user and pass:

```
nessus -m -q 127.0.0.1 1241 user pass
```

If the Nessus client and daemon are on the same server, this command can be modified using the -x option to disable certificate checks:

```
nessus -m -q -x 127.0.0.1 1241 user pass
```

Otherwise, Nessus may ask for guidance as to how it should attempt to encrypt the access between the Nessus client and daemon, with the following message:

```
Please choose your level of SSL paranoia (Hint: if you want to manage many
servers from your client, choose 2. Otherwise, choose 1, or 3, if you are
paranoid).
```

Choosing 2 allows one Nessus client to access multiple Nessus daemons as long as their username and password are known. Options 1 and 3 are for PKI-style deployments of Nessus daemons and are not covered in this article. The Nessus client will continue to prompt the user to verify whether a specific certificate is correct.

Normally, with a GUI tool, the GUI handles selection of which vulnerability checks to run and how the Nessus daemon should perform. This information is stored in a hidden configuration file named .nessusrc and is located in your home directory. Whenever a scan is run (from the GUI or from the command line), this file is overwritten. Without a GUI, though, invoking the Nessus client with the -m –q option is the right way to make the default configuration file for Nessus.

For a set of networks or IP ranges to be scanned, they must be specified and listed in one file. To specify an IP address, simply list it. To specify a range of addresses, CIDR blocks and ranges can be used. All target IP addresses need to be concatenated onto one line and should be saved into a file for use by the Nessus client. The following lines are example valid target IP specifications:

```
10.10.20.23
10.10.20.0/24
10.10.20.40-33
10.10.20-30.40-45
```

That last example would scan each class C network from 10.10.20.0/24 to 10.10.30.0/24, but would only scan hosts .40 through .45 within each network.

For our example, let's call the file to put this information into the "targets.txt" file. The Nessus client will need to know this, and which file it should output its data to.

```
nessus -q 127.0.0.1 1241 user pass ./targets.txt output.nsr
```

This will launch a scan against the systems contained on the first line of the targets.txt file. Watching the processes for nessusd while scanning is ongoing reveals what the particular instance of nessusd is attempting to do. For example:

```
nessusd: testing 127.0.0.1 (/usr/local/lib/nessus/plugins/roads_cgi.nasl)
```

could be a typical test.

While the Nessus daemon is active, a trace of its progress can be found by watching its logfile:

```
tail -f /usr/local/var/nessus/logs/nessusd.messages
```

Once the scan is completed, the Nessus client will write a report as shown in the example below:

```
127.0.0.1|http (80/tcp)|11408|INFO|;The remote host appears to be running a
version of;Apache 2.x which is older than 2.0.43;;This version allows an
attacker to view the source code;of CGI scripts via a POST request made to a
directory;with both WebDAV and CGI enabled.;;*** Note that Nessus solely
relied on the version number;*** of the remote server to issue this warning.
This might;*** be a false positive;;Solution : Upgrade to version 2.0.43;See
also : http://www.apache.org/dist/httpd/CHANGES_2.0;Risk factor :
Medium;CVE : CAN-2002-1156;BID : 6065;
```

The Nessus client defaults to writing an ASCII-style report with a relatively straightforward documentation of the vulnerabilities and hosts discovered. Other report formats, including HTML, a one-line style, and even XML, are also available.

Reports can also be converted on the fly as needed using the Nessus client. For example, to convert the output.nsr file from our example to an HTML file, the following command would be used:

```
nessus –i output.nsr –o output.html
```

In the example above, the output.html file could be served by any Web server, or loaded into any Web browser.

Using the X Windows GUI

The Nessus client, if compiled with the GTK toolkit, also provides a GUI for scan configuration and analysis of the discovered vulnerabilities. The GUI allows the Nessus user to select which Nessus daemon will be used for testing, which vulnerabilities will be used during the test, and how the Nessus daemon will perform the vulnerability scanning process. Many of these options are discussed in the next section.

The Nessus GUI can also be used to evaluate the results of a vulnerability scan. Information discovered during a vulnerability scan can be sorted by network addresses, DNS domain names, IP addresses, open ports, and discovered vulnerability types and severities. The GUI can also be used to save and convert Nessus reports in a variety of formats.

Tweaking Nessus: The Nessusrc File

Nessus's performance, accuracy, and intrusiveness can all be controlled from the nessusrc file. When the Nessus GUI client runs, it creates the nessusrc file with the options selected by the Nessus user. If no options are specified by the Nessus client, the Nessus daemon will choose a variety of its own default values.

To build customer nessusrc files, an easy technique is to first run a basic scan with the Nessus GUI client, and then to use the generated nessusrc file as a template that can be modified. The default nessusrc file will be located in the user's home directory as a hidden file named .nessusrc. For example, a root user would find their nessusrc file in /root/.nessusrc.

The nessusrc file has several sections. These are SERVER_PREFS, SCANNER_SET, PLUGINS_PREFS, and PLUGIN_SET. The SERVER_PREFS controls a variety of settings that relate to how the Nessus daemon performs scanning. The SCANNER_SET and PLUGIN_SET sections identify which NASL scripts are enabled and disabled. The PLUGIN_PREFS section provides information, such as a desired SNMP community string, which may be used by one or more NASL scripts.

Several keywords in the SERVER_PREFS section are useful in configuring a Nessus scan. Here is an example SERVER_PREFS section from a nessusrc file:

```
begin(SERVER_PREFS)
port_range = 1-1024
optimize_test = yes
safe_checks = yes
max_hosts = 30
max_checks = 10
cgi_path = /cgi-bin:/scripts
report_killed_plugins = yes
language = english
per_user_base = /usr/local/var/nessus/users
checks_read_timeout = 15
delay_between_tests = 1
non_simult_ports = 139
plugins_timeout = 320
auto_enable_dependencies = yes
use_mac_addr = no
save_knowledge_base = no
kb_restore = no
only_test_hosts_whose_kb_we_dont_have = no
only_test_hosts_whose_kb_we_have = no
kb_dont_replay_scanners = no
kb_dont_replay_info_gathering = no
kb_dont_replay_attacks = no
kb_dont_replay_denials = no
kb_max_age = 864000
plugin_upload = no
plugin_upload_suffixes = .nasl
save_session = no
save_empty_sessions = no
host_expansion = ip
reverse_lookup = no
detached_scan = no
```

```
continuous_scan = no
unscanned_closed = no
diff_scan = no
log_whole_attack = no
slice_network_addresses = no
end(SERVER_PREFS)
```

To limit or increase Nessus's activity (and required resources), the keywords max_checks and max_hosts can be modified. Increasing max_checks will allow the Nessus daemon to launch more processes per tested system, and increasing the max_hosts variable will increase the total number of systems a Nessus daemon will check at once. On large servers with extra memory, bandwidth, and CPU resources, a Nessus scan may run much faster with higher values than the defaults.

There is no exact formula for tuning a Nessus scan for optimized performance. However, during a scan, if CPU utilization is low, increasing these numbers by 50% may decrease the amount of time to complete a scan. Also, if the desired scan is not a full scan, but just a quick check for a few NASL scripts, increasing the max_hosts value may decrease the overall scan time.

The optimize_test keyword invokes logic at the Nessus daemon such that it will only attempt to launch a vulnerability check if a dependency has already been discovered. For example, with this option enabled, a "writeable anonymous FTP directory" check will not get invoked unless the "anonymous FTP access" check has returned positive. Similarly, the auto_enable_dependencies keyword can be used to enable a plugin that is required during testing. In the above example, if a Nessus user attempted to do a "writeable anonymous FTP directory" sweep of a network but forgot to enable the "anonymous FTP access" check, the scan would not return meaningful results. This keyword allows Nessus users to perform quick checks without having to memorize the potentially intricate dependencies some Nessus NASL plug-ins have.

The safe_checks keyword allows the Nessus daemon to select code within some NASL scripts that is less intrusive but possibly more prone to false positives. For example, a particular NASL script to look for a buffer overflow could attempt to actually flood an application with potential exploit data, or with safe_checks enabled it might read the banner information and make a decision about the vulnerability based on that.

If the particular scan includes a port scan, the ports to be probed are indicated with the port_range keyword. Ports can be listed individually or in ranges separated by commas. For example, 22,25,80,443,31330–31430 would scan the default ports for SSH, SMTP, HTTP, HTTPS, and all ports between and including 31330 and 31430.

In addition, the unscanned_closed option causes the Nessus daemon to treat ports that were not scanned as if they were closed. This is advantageous for doing "point" scans. For example, when using a Nessus scan to find SMTP servers on port 25, NASL scripts that perform checks on ports other than 25 would automatically not be invoked.

The cgi_path keyword can be used to optimize certain NASL scripts that attempt to find vulnerabilities on Web servers with CGI-BIN paths in non-obvious places. This variable can be modified if the Nessus user has knowledge about how a particular Web server has configured its CGI-BIN location.

The last keyword in the SERVER_PREFS section is the slice_network_addresses command. When enabled, the order in which IP addresses are tested is randomized. In an environment which is not reactive to a Nessus scan, this has no effect on results of the scan. However, some dual-homed servers may experience several full scans at the same time and suffer some sort of negative impact. Consider a router, switch, or even a database server which is multi-homed on more than one network. If such a system were hit on each interface at the same time, it might experience slower network response time, slower server response times, or possibly even some sort of denial of service.

The PLUGIN_PREFS section specifies a wide variety of options for the enabled Nessus plug-ins. Some of these can be used to increase the accuracy and number of vulnerabilities discovered on any network. For example, Nessus can also be preconfigured with additional information, such as an SNMP community string and Windows network "auditing" account information. This information can be used by the Nessus daemon to actually log on to the tested hosts and check specific settings.

In the case of SNMP, several dozen Nessus checks make direct queries to the SNMP information for security information. To specify an SNMP community string to be used during testing, the SNMP port scan[entry]:Community name : = entry would be coded with the desired string. It is important to realize that there still may be other NASL scripts which do SNMP checks that do not take advantage of this setting, but that there are one or more checks that do. In the specific case of SNMP, there is a specific NASL plugin that attempts to brute-force several common SNMP community names. However, there are several dozen other NASL SNMP checks that use any discovered SNMP community names, and any preconfigured in the nessusrc file.

For Windows auditing, if a Primary Domain Controller is being used, then a global account which has read-only registry rights to each server in the domain can be used by Nessus to actually log in to an evaluated system. This account's username, password, and domain are placed in the nessusrc Login configurations options, specifically, within the PLUGIN_PREFS section. These entries are shown below:

```
Login configurations[entry]:SMB account : =
Login configurations[password]:SMB password : =
Login configurations[entry]:SMB domain (optional) : =
```

When Nessus sees that a target host is a potential Windows host, it will attempt to log in to the system and make registry calls to determine the configuration and whether any vulnerabilities exist.

(For more information, see "Utilizing Domain Credentials to Enhance Nessus Scans," by Ty Gast of the Security Assurance Group, located at *http://www.nessus.org/doc/nessus_domain_whitepaper.pdf*).

To determine whether a host is alive, the following settings can be configured for Nessus:

```
Ping the remote host[entry]:TCP ping destination port(s) : = built-in
Ping the remote host[checkbox]:Do a TCP ping = yes
Ping the remote host[checkbox]:Do an ICMP ping = no
```

If either a TCP ping or an ICMP ping check is enabled, then one of those checks has to return positive, or else the tested IP address will be assumed not to be active. This is important when testing systems that are protected by firewalls.

From outside a firewall, or with the presence of host-based firewalls, this may be the only way to accurately test a host's vulnerabilities. Consider a host serving a SQL database on a high port that also has a firewall in front of it which prevents any access to the server except via SQL. Any attempt to ping it with an ICMP packet, make a TCP connection on any port other than the SQL port, or send any UDP packet to it would not be successful. However, if SQL were running on the default SQL port, the first NASL plugin that connected to it would succeed and the vulnerabilities would be tested.

There are almost 2000 unique Nessus NASL scripts available.

When scanning a large network, choosing either ICMP or TCP host enumeration is recommended. Otherwise, each NASL plugin script must time out on its own. This makes for a much longer, but also much more accurate, Nessus scan. For a TCP ping, Nessus will attempt to connect to several common ports in the 1–1024 range. If desired, a customer port list can be placed in the TCP ping destination port(s) configuration option.

There are several hundred more settings for the PLUGIN_PREFS section. Readers interested in crafting custom settings for other variables are encouraged to experiment with the Nessus GUI client and observe the changes made in the produced nessusrc files. Readers are also encouraged to consult the Nessus mailing list archive highlighted at the end of this article.

The last two sections in the nessusrc file are PLUGIN_SET and SCANNER_SET. Both of these sections list which NASL plug-ins are enabled and disabled. The SCANNER_SET section is used to identify plug-ins that perform some sort of port scanning. These are executed before the regular vulnerability-checking plug-ins. Historically, Nessus has been able to invoke a wide variety of port scanning and host enumeration tools, such as NMAP (*http://www.nmap.org*) and SNMPWALK. Currently, Nessus 2.0 does not require any external port scanning tools. When listing a plugin for testing, the ID will be printed followed by a simple "yes" if it is to be considered for testing, or a "no" if it is to be excluded. Here is an example excerpt from a nessusrc file:

```
10909 = no
10330 = no
11268 = no
11480 = yes
10351 = yes
10010 = yes
10536 = yes
10440 = yes
11210 = yes
11641 = yes
11554 = yes
```

Nessus Vulnerability Checks and NASL

There are almost 2000 unique Nessus NASL scripts available. These scripts are each labeled with a unique Nessus ID, contain a brief description of the relevant security audit or vulnerability, and also have links to CVE (*http://cve.mitre.org*) and Symantec's BugTraq (*http://www.securityfocus.com*). Each NASL script is grouped into a family that focuses on specific services, operating systems, and "problem areas" such as peer-to-peer (P2P) file sharing.

Many of the vulnerability checks performed by NASL are unique tools unto themselves. For example, the traceroute tool includes logic to perform multiple traceroute checks to many hosts without colliding. There are checks that identify wireless access points by their TCP/IP operating-system fingerprint, individual Web interface, and SNMP or FTP banner information.

Nessus includes plug-ins that can perform extremely complex audits of Web services. For example, the "torture" CGI NASL script actually reads in the values of the Web forms automatically found during a Nessus scan, and commonly queries these forms with typical CGI-BIN exploit techniques. Similar checks are performed by commercial Web-application testing tools.

Deploying Permanent Nessus Daemons

When deploying Nessus, many organizations take advantage of placing a server permanently within an infrastructure that is dedicated to scanning. These servers are often located within server farms, behind firewalls, and within DMZs. Sometimes they are also deployed permanently outside of a network to perform an external audit.

The advantage to deploying dedicated servers is that they are always on and ready to perform a scan from the perfect location. Anyone who has performed a network security audit and has had to drag a laptop around a network to get to the right spot to perform a scan knows what a hassle it can be. By pre-placing a Nessus within a network, the time it takes to perform a scan is just the time it takes to perform a scan. There is no more need to walk to the building that has the firewall in it, fly to the remote location, or find the proper network cable or switch port to plug into.

The systems that run the Nessus daemons should be secured so that no remote services are running except those that are required to securely communicate with the device. For example, a common "hardened" Nessus daemon server would include the actual "nessusd" process, which normally listens on port 1241, and the Secure Shell daemon, which listens on port 22. It is also common for users to use a system firewall to limit connections to the Nessus daemon from the authorized network users and to limit access to port 22 from only "trusted" IP addresses within the network.

When a Nessus daemon is deployed, very little vulnerability information will be present on the device. Most scan data passes through the Nessus daemon and is not stored there.

Minimizing Scanning Impact

Scanning with any network vulnerability scanner, including Nessus, can have a negative impact on the target network. This impact can come from an overall slowdown or degradation in network performance, or a denial-of-service event on one or more devices.

When Nessus 1.0 was originally available, its standard tests included a list of DoS tests that were enabled by default. Often, a Nessus user in the late '90s would end up finding out very quickly if their network systems were vulnerable to the "ping of death" DoS attacks and several others. With Nessus 2.0, great care has been taken to limit the number of potentially damaging DoS checks that Nessus can perform. DoS checks are no longer enabled by default.

When Nessus performs its checks, the traffic that it generates may have a negative impact on a switch or firewall. Not only does the scan send packets across a network, but often, one packet in a port scan will be considered a new session by a switch or firewall. For example, some appliance firewalls that keep track of unique network sessions will quickly lose track of their current valid sessions when faced with a large port scan. If a firewall can only keep track of 65,000 unique network sessions, a port scan of one host can quickly fill this buffer up, pushing out valid sessions. Similarly, poorly designed firmware in routers, switches, and some firewalls doesn't handle the large numbers of network sessions generated during a vulnerability scan. The flaws in the handling can be very subtle.

For example, the author used to work for a certain routing vendor, and one of our products would reboot when scanned with a certain commercial vulnerability scanner. Duplicating the same scan with Nessus or NMAP did not cause the reboot, and only doing a low-level packet trace resulted in identifying the exact sequence of events which brought the network device down.

To avoid stressing the network infrastructure, Nessus daemons should be placed as close as possible to their desired targets. Any deployment that minimizes the flow of host identification, port scans, and vulnerability checks across the core routers and switches should be considered. Also, when doing a vulnerability scan for the first few times on a large enterprise network, attempt to correlate the scan with the CPU load, packet rates, and any other statistics that can be obtained from network engineering. These statistics may indicate faults in the network before actual faults are caused, if any.

Finally, when a specific host is identified, conducting vulnerability scans on the host may cause some of its applications or the entire operating system to crash. Even with careful writing and testing of NASL scripts, and with careful laboratory testing, there is no way to guarantee that an NASL script will have no impact on every possible system and application. All vulnerability scanners have this problem. Even if there were a way to perform regression testing on all known operating systems and applications, there are still thousands of custom applications that the Nessus project does not have access to test.

When testing a production network, a lab test on a certain group of NASL scripts can be used as a benchmark to predict the impact of running the NASL scripts on the entire network. If a network has many similar hosts, such as a large network of Red Hat Apache servers, doing some quick tests in a lab before blasting the entire network can give a sense of the impact that will result. However, in networks with extremely diverse families of servers, it is impossible to predict the impact of a scan without actually doing the scan itself.

Unlike other vulnerability scanners, Nessus does attempt to alert the user if it may have crashed a server during testing. It does this by looking for changes in the available ports of a tested server. If during a port scan, it sees that ports 80, 443, and 6000 are open, it will alert the user that the server (or service) may have crashed if one of the ports goes away.

Readers are advised that a poorly run network may crash for many reasons, including failing to survive a vulnerability scan. They should not be dissuaded from performing active scans, but should realize that doing any network activity can impact the network. Most vulnerability scans simply exercise systems that have not been used in a long while, or "operational" systems that were not fully tested.

The Future of Nessus

The Nessus project is currently funded by Tenable Network Security (*http://www.tenablesecurity.com*), which is a commercial network security product company.

Although the Nessus project continues to be open sourced, Tenable has introduced several commercial products that greatly enhance Nessus for large enterprises.

Long-term plans for Nessus include a preliminary design of Nessus 3.0, which will continue to make Nessus one of the top vulnerability scanners available to the network security community. In the short term, the Nessus Web site recently added the ability for users to submit information privately and anonymously about which of their NASL scripts have produced false positives for them. This helps Tenable and the Nessus project quickly identify when a NASL script needs to be modified. Nessus 2.0 was released early in 2003, and ports to the OS X operating system have been completed.

There is a wide variety of information about Nessus available at *http://www.nessus.org*, as well as a searchable mailing list located at *http://msgs.securepoint.com/nessus/*.

coordinated incident response procedures

A Case History

by Dario Forte

Dario Forte, after working for the Italian government for 15 years, is now security advisor for the European Electronic Crime Task Force (EECTF), a nongovernmental group dedicated to incident response. He is also the founder of Incident Response Italy, developed at the University of Milan, where he teaches incident response management.

dario.forte@acm.org

Background

In mid-2002, two groups of malicious hackers were identified by the Italian Financial Police as being responsible for a series of attacks on over a thousand targets throughout the world. The backtracing procedure was seriously complicated by the fact that the groups used numerous stepping-stones and camouflage techniques, such as IPv6 tunneling. In this article we take a general look at the attack methods and illustrate the techniques and steps used to backtrace them. For reasons of privacy we will not name the targets but will use letters instead:

A – The German target used as an initial stepping-stone to attack the American governmental sites indicated below

B – The main American governmental target attacked by the group

C – Another American governmental target attacked by the group, which served as the starting point for the investigation

SS1 – A university machine used as a repository to hide rootkits and other tools used in the post-intrusion phase

A Coordinated Attack

In September 2001, the owner of C realized that one of his machines (an IRIX) had been attacked. The exploit had been launched from a German machine, which had previously been compromised by an exploit from its resident Web server. The system administrators for C later reported that commands had been sent from the German machine to download certain post-intrusion tools (including rootkits) from a third machine, SS1, located at an American university. Here is the general scheme.

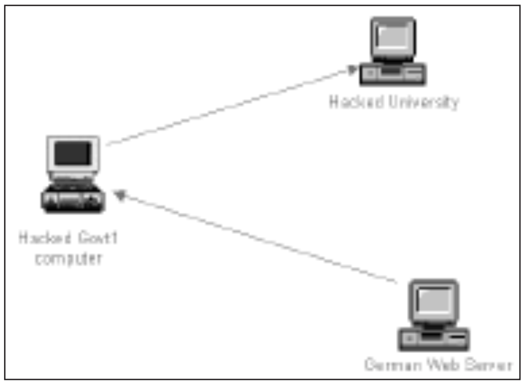


Figure 1

Reconstructing what happened to C was possible thanks to the presence of an IDS that monitored the target. While this did not permit a response in realtime, it did make it possible to recover a series of logs that illustrated what had happened. The logs were usable because they were not on the attacked machine. In the meantime a post-mortem exam was carried out on the German machine, A, that had been used as a stepping-stone. The method used to compromise the German machine was generally conventional, but had a number of personalized touches added by the attackers.

Requirement No. 1: Reconstruct the Events

One of the first steps in this sort of investigation is to check how much time passed between the last update of the machine and the attack. This may help identify the exploit that was used to achieve the intrusion. In this case it was a Linux machine that was not running the latest release. At the time of the intrusion, the bug exploited to compromise the box was the then-known wu-ftpd site command exploit.

An attacker who penetrates a machine installs a rootkit in order to keep the machine compromised. One of the attack group's characteristics was the use of completely self-made rootkits along with materials known to the security community. The choice depended on the type of target (in the case we are examining, there were nearly a thousand boxes compromised worldwide).

The t0rn rootkit was chosen for the German machine and installed in /usr/info/.t0rn. Evidently, the reasoning used in this case was, given that the system was generally poorly administered, it was not worthwhile to keep it compromised with something exotic, since it was unlikely that the administrator would realize what was going on anyway. In any case, it may be helpful to consult /usr/src/ to try to determine what is going on. In the specific case, a lot of information was found in /usr/src/.r00t.

t0rn rootkit, like most tools of its ilk, is configured to hide certain network connections. This is the usual syntax found during a post-mortem (the IP addresses are fictitious):

```
65.93.*.*
195.242.20.*
```

where * is used to hide all the addresses of the block. The first instance usually occurs to hide classes of dynamic IPs that the attacker has access to (e.g., ADSL and dialup based on DHCP). In the second instance, an entire class is indicated, including one or more machines compromised for the long term by the attacker.

Once the groundwork is laid, the hacking tools are installed. The choice of tools is completely up to the attacker. In our case, the hacktools installed were:

`7350wu`	exploit to hack into the wu-ftpd (used on this system, too)
`massroot`	exploit for a bug in the telnetd of IRIX systems
`statdx`	exploit for the rpc.statd of RedHat
`mirkforce`	attack tool to disrupt IRC communication
`papasmurf`	smurf attack tool, a denial-of-service tool
`seclpd`	exploit for lpd in RedHat 7.1

Please note that we are talking about an attack that occurred in 2001. Interestingly, several text files made clear that A was the machine used to attack the governmental sites. The attacked networks, in fact, were in the 136.*.*.* and 137.*.*.* nets. It might be useful to seek subdivisions by operating system in these files. In this case FreeBSD, IRIX, Linux, and SunOS were found, along with a ".txt" file which contained progress info of the scanning.

This proves how important it is to correlate what is found on one machine with what is found on the one that appears to be directly connected to it. The correlation, especially if done on more than two machines, can map out the events with a certain margin of certainty and point back to a single source.

Another item usually installed is psyBNC IRC Bounce BOT. In this case it was used as a deflector to participate in IRC communication without revealing the hacker's IP number, thus avoiding DoS attacks on the hacker's machine. Usually the attacker installs BOT with the his IRC nick, which, in many cases, turns out to be very important for final backtracing.

The following presents the main steps taken by the attacker after the intrusion:

1 System is penetrated through an exploit.
2 A rootkit is installed.
3 nc-ftpd is installed.
4 A port scanner is installed.
5 A sniffer is installed.
6 A psyBNC BOT is compiled and installed.
7 A rootkit is fine-tuned.
8 A file with IP numbers is created.
9 The "real use activity" starts.

The compilation of the items downloaded by the third machine (SS1 in this case) is generally carried out either on the attacker's machine or directly on the compromised machine. Both choices have their pros and cons. For example, compiling on the attacker's machine might speed things up but risks instability due to potential differences in platforms. On the other hand, one cannot be sure that there is a compiler on the compromised machine, even though it is quite probable for relatively simple cases.

Further Correlations

In the case in question, there was another positive factor for the investigation: cross-checking of the SS1 machine. When a machine is used as a repository for tools that will be downloaded onto target machines, it may happen, with a bit of luck, that additional cross-references can be found to correlate all the necessary information. Given that most such "containers" are located on university networks, we find ourselves confronted with the following situation:

- University officials provide system logs and an image of the compromised computer.
- The compromising of the US university machine is linked to the compromised third-party computer.
- The university computer is used as a "toolbox." All links between the .edu computer and the real target require a physical-level search that, very often, reveals a dialup connection.
- A proper HD analysis can uncover the intruder's rootkit.

This check, in the specific case, allowed the real departure-point ISP of one of the attackers to be backtraced. The same control also provided several important correlations regarding attacks on B, from which important sensitive files were stolen. Figure 2 diagrams the basic correlation.

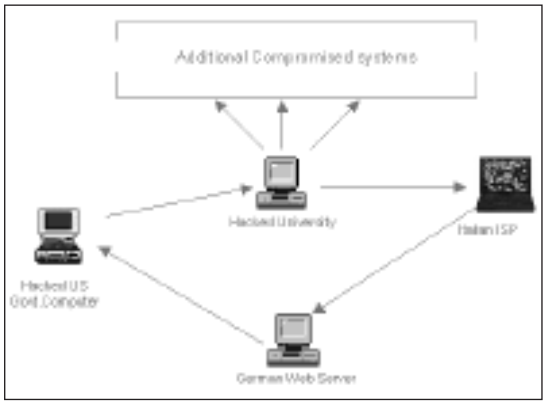


Figure 2

Conclusion

Another successful aspect of the investigation was that all the investigators spoke the same technical language. Terminology, log type, image format, tools, and PGP keys were agreed on before beginning the investigation, proving the fundamental importance of setting things up well before getting started.

We have only touched the tip of the iceberg in this article and discussed only those parts of the investigation not protected by nondisclosure restrictions. The investigation was anything but simple. The operation, known as "Rootkit," took more than one year and involved five European and American investigative agencies (military and civilian). Fourteen people were charged, including four minors. Most of them worked as security consultants or managers in large multinational companies. More than 40 computers and almost one terabyte of data were seized, along with thousands of CD-

ROMs and DVDs. Many credit card files were recovered. If it had not been for the close international collaboration, it might not have been possible to track down the perpetrators of over 1000 worldwide attacks, who were so active and so skillful as to be able to write their own rootkits and log wipers, used on the most "important" machines, and so crafty as to use IPv6 tunneling. Unfortunately (or fortunately), it's a small world: Some of the people charged as a result of this investigation had also punched holes in a Mexican honeynet, going so far as to get into the Honeynet Project's famous "scan of the month."

Acknowledgments

I would like to thank Eddie Autelli, Dave House, Alvin Allen, Kevin Manson, Troy Bettencourt, and Chris Fischer, all members of EECTF, for all the incredible support I have received over our years of working together.

how are we going to patch all these boxes?

by Ryan Russell

Ryan Russell is an independent security expert, author, and speaker. Presently, he is contracting at BigFix, Inc. where he is helping them expand their UNIX support.

ryan@thievco.com

Chances are excellent that I don't need to point out to you that you've been beset by worms. If you have a sizable network and run Microsoft Windows, you have probably seen the front lines of the worm battle. If you run a Microsoft-free environment, you at least get to see the fallout in terms of constant probes and email-borne malicious code (or bounces, when the worm decides to be your email address for the day).

Are worms strictly a Windows problem? Technically, no. There have been worms for most of the major operating systems, and that includes most of the UNIX-like flavors. For all practical purposes right now, however, worms are a Windows problem. I happen to be one of those people who thinks that this has primarily to do with market share rather than any particular technical reason. Call me when Linux has had a 90%+ market share for a couple of years, and I'll be more than happy to revisit my opinion. I just don't think that the malicious code authors will give up and find something useful to do simply because the UNIX security model is "too hard."

That said, this article is about patching, rather than worms themselves. Obviously, the worms are the driving factor behind the recent interest in keeping patches up-to-date. This doesn't mean that your non-Windows operating systems aren't just as deserving of having their patches kept current, far from it. It's just that in terms of fire-fighting, you tend to go after the towering inferno first, and worry about the kitchen fires later. As we'll see, though, maybe while you're figuring out patching, you can take care of *all* your computers, and save some future headache.

Let's review. Here's the list of things that you're supposed to do that will "help keep you secure":

- Install a firewall
- Don't run unnecessary services
- Install patches
- Run antivirus (AV) software and keep it up to date
- Use an intrusion detection system (IDS)
- Educate users

Does this list look familiar? I think it's probably something like 15 years old now (with the addition of more recent developments like IDS, of course). Do you do all these? Of course you don't; at least, you don't do them right. It's not your fault, you can't. You've got nowhere near enough budget or management support. The closest I've seen to this being done right is in the military (well, certain military) networks. I don't doubt that I'll get email from net/sysadmins at these facilities disabusing me of that impression, too.

Sure, you've got the easy parts "done." You've got the firewall, and an enterprise AV software license, and the IDS is running. Maybe you've taken a stab at user education. How about the patches and disabling of unneeded software? Is the AV software actually installed, running, and up-to-date everywhere?

The basic pattern is that the pieces that are few, central, and under your direct control (firewalls, IDS, central servers) are well managed. The pieces that are many, distrib-

The challenge is putting patches on and keeping them on.

uted, and under user control are poorly managed. This is not surprising. With the central devices, you can do amazing things. For example, count how many you have. You can physically lay hands on them. You can lock them up and keep other people from laying hands on them.

Sadly, the centralized pieces, the ones you can actually get a handle on, have become less effective. Firewalls seem a lot more porous in the past few years. Internet applications have become very adept at traversing various types of firewalls. You've got computers coming and going, portable computers, VPN links, wireless access points (authorized or not). The two most prevalent worm attack points, HTTP and SMTP, were the very first two to be enabled at the firewall when it went in.

Signature-based technologies (namely, most AV and IDS products) have one basic drawback: The threat must be known in order for it to be caught. Generally speaking, in order to create a signature for something, you have to have caught and analyzed that something. This means that AV always lags behind the malicious code, and IDSes almost always lag behind the exploit. It has been reported that the SQL Slammer worm spread to the majority of vulnerable hosts in as little as eight minutes. That's not quite enough time to capture the worm, analyze it, create a signature, distribute it to customers, and have them put the new signature into production. That's an extreme case, but it illustrates the basic point. Keep in mind also that IDSes generally are completely passive; they only detect. Even if you could detect the problem within minutes, that wouldn't solve it. However, knowing that it is going on is the first requirement for fixing it.

Please don't take that to mean that I'm saying these technologies are useless, or that you can quit buying them. Not at all. They are still very much your minimum price of admission for installing an Internet connection. Sure, the IDS signature may not be available for a day after the worm is launched, but how else are you going to know that the worm has been running on your network since yesterday and is still going strong? You're going to need it to tell you which inside boxes are infected. There is always the option of using a packet capture program and manually reading through the packets. You should be capable of doing so when needed, but your IDS is supposed to automate part of this process for you.

What I am saying is that this (relative) weakening of the perimeter, coupled with the increased threat of the upsurge in worm volume, means that the level of security of your inside machines must increase. The word "security" means, in the context of worms, installing patches. Not all worms utilize only known vulnerabilities to infect a host, but the majority of them do, and it tends to be the most effective vector. One of these days, we will see a worm that uses an unknown vulnerability (no patch available ahead of time, a.k.a. a 0-day worm), but by definition, there is no patch to put on before it hits, so we won't worry about that for this discussion. (The general mechanism I talk about is still helpful in the case of a 0-day worm for clean-up, though.)

So, the challenge is to apply the various security patches and, secondarily, a little host hardening and monitoring. I further qualify that: The challenge is putting patches on and keeping them on. Why is keeping them on a challenge?

Imagine this scenario: A new Microsoft security hotfix is available. You mentally add it to the to-do list, until you get wind of a new worm that takes advantage of that vulnerability. So, you log in as domain admin and run your script:

```
C:\admintools\domainrun mydomain c:\dl\q8675309.exe
```

For purposes of discussion, this fictional (though quite feasible) tool finds all hosts in the domain "mydomain," uploads the program given, and runs it. Are we done?

Of course not. Here's a partial list of things that might have gone wrong:

- Some hosts were not in the domain.
- Some hosts were not on at the time.
- Some hosts were on, but not on the network at the time.
- Some hosts were Win9x, and the patch was for NT, 2000, and XP.
- Some hosts have the service that allows the remote install turned off.
- Some hosts didn't have a new enough service-pack level for that hotfix.
- Some hosts were running another language version of Windows and needed a slightly different hotfix.
- On some hosts, the patch program crashed.
- Some hosts were patched, but then configuration was changed later, and the patch got downgraded.

The list goes on, and that's just the Windows patches. For non-Windows, you can substitute "root password was changed" for "not in the domain," or similar. The concept and problems still apply.

You might imagine running the same command periodically, which fixes some of the problems. You might think about including similar commands in your login scripts. Either of these looks somewhat workable, until you've accumulated a couple of hundred patches and realize how long it is going to take for each cycle.

Having stated what I think the current set of challenges is, allow me to discuss how I think the challenges can best be met. In the interest of disclosing my biases to the reader, I want to point out that as of this writing, I'm working as a contractor at a company that sells enterprise software that does patch management (among other things). A lot of my research comes from having seen how that software works, and what I'm going to suggest you do looks a lot like what they sell. However, I believe they are on the right track and that the solution is valid. They aren't the only ones who sell software like this, so please take my statements as general and make your own decision about how you're going to get a system like the one I will describe. I'm not here to make a sales pitch.

In order to continually determine whether a patch is needed on a particular host, you will need a piece of software continually running on that host. I'm going to call this piece of software an "agent," but you can substitute whatever word you like. This agent must periodically check the system to see whether the host has a particular patch. Note that it is checking, not applying (yet). The checking process should be much quicker, overall, and therefore tolerable to do frequently. The agent will do this for each patch it knows about. Therefore, there must be some sort of list of all existing patches you wish to worry about, and must be updated as new patches are made available. I'll refer to this as "content." To recap, there is an agent that runs on every host, and based on some content it is given, it can determine which patches are in place and which are not.

Astute readers will be asking themselves how the agent gets onto the host in the first place. Ah, so there is a bootstrapping problem? Yes, and it's an ongoing bootstrapping problem, because new hosts get added all the time, and old hosts will somehow manage to lose their agents on occasion.

Allow me to briefly discuss the bootstrapping issue without going into great detail. There are a number of ways to do this – there's no single "best" way – and you may choose to use some combination of techniques. First, you could manually visit each host. It's boring, but at least it's obvious. Second, the techniques that I derided as having many problems for delivering a patch are actually not a bad way to get your 80% of hosts covered in a hurry. Use your domain credentials to blast the installer out, either with a command line program or by using the login script facilities. Third, you need some way to monitor your network for agentless hosts. You might do this with network scanning (say, the agent runs on a particular port and has a particular banner), you might tie in with your DNS/DHCP infrastructure, or you might ask your routers and switches which hosts they know about. I recommend all of the above.

Once you've got your agent running everywhere, what does it do exactly? Well, a good agent can do anything you like, but I'm here to talk about patching. You want the agent to download and install the patch. What, automatically and with no intervention? No, of course not. Don't run screaming quite yet, I'm not talking about a completely automatic, blind-faith patch install. I'm talking about putting on the patch that you will put on eventually, but at a schedule you pick, and without having to trek all the way to every host. The way we do this is by having all the agents report in to the central console.

In our design, the central console serves several functions:

- It tracks all the agents (so we know when they disappear).
- It reports which hosts need which patches (with appropriate sorting and searching functions).
- It issues commands to the agents to perform a particular patch install (at the prompting of its human operator).

Clearly, there is much more that the console could do (e.g., statistics, history, inventory tracking), but I'm going to stick with discussing these base functions.

As a console operator, your job is to look at what patches are needed and to apply them. After the patches are applied, the central console will report back on whether those agents now indicate their hosts are patched. Can you get away with doing Select All and clicking Go? Not if you're like most organizations. Many organizations have to deal with change control, patch program quality concerns, breaking software compatibility, service-level agreements, and similar issues. Well, no problem, your patch management agent is still going to make your life easier.

What does your change-control procedure say you should do to approve a patch for mass deployment? Test on your testbed hosts? Select those, and click Go. Test on some portion of your network? Select your IT department desktops, or your least favorite users, or whatever subset you like. You might even try finding willing volunteers, to avoid impacting your own long-term employment. Click Go. Want to make sure the patch is rolled out when help-desk personnel are available? Schedule according to day, hour, geography, follow-the-sun, whatever, click Go.

Have a set of hosts that your vendor supplies as "black boxes," even though they are running a stock version of Windows that needs to be patched like the rest, and they refuse to support you if you patch yourself? Well, sorry, can't help you there, don't click Go. They probably wouldn't let you put the agent on in the first place.

And, of course, when you've neglected your rollout schedule of patches for whatever reason, when the day rolls around that a worm is released for a vulnerability that you haven't fully vetted the patch for yet, you can decide if you'd rather have the worm or the patch of unknown quality. At least, if you decide the latter, you've now got the means to implement that decision in a hurry. (I would not recommend that admins not follow change-control procedure, or that they have faith that their vendors always produce flawless patch software, but most of the time you'd rather not pick the worm.)

So, is everything perfect? As described, such a system will let you determine which patches are needed, apply them according to your schedule, and verify that they stay there. This meets all of our requirements.

An actual implementation always leaves things to be desired. Could such a system lessen security in any way? Just like any software, it must be written with security in mind, specifically meaning, no security holes. Yes, if your agent has holes, then you've added that many more holes to each host. At least if these are discovered in the agent, you've got an easy way to upgrade the agent software in a hurry. (Actually, self-upgrade is not necessarily an easy problem, so if you're investigating such software, make sure to check into that.)

So, our agent is bug-free; what else can go wrong? Can an attacker pretend to be the console, and tell all the agents to download and install backdoor.exe? One would hope not. One solution that seems reasonable is to use public key cryptography to issue commands. In other words, PGP/GPG-sign the commands at the console and have the agents verify the signature against a stored key. And don't forget to deal with replay attacks (check that old, saved, signed commands won't work later).

And, obviously, keep careful control of the console. The public key crypto will help against some attacks, but if an attacker has full control of the console over a period of time, including installing a keyboard sniffer, it may not make any difference.

My hope is that this article has made you think a bit about some of the practical steps you can take to help keep your network worm-free. No solution is perfect, but we should still look for tools that can make a significant impact. The tools I've discussed here would help maintain a base level of security against worms and script kiddie-level attacks. Simple patching may not help save you from a clever, determined attacker, but who has time to worry about them with all these worms on our networks?

REFERENCES

"eWEEK Review of Patch Management Solutions," *eWEEK*, *http://www.eweek.com/article2/0,3959,1246104,00.asp*.

"PatchLink Helps Keep Windows Closed," *Network Computing*, *http://www.networkcomputing.com/1318/1318f3.html*.

"Windows Patch Management Tools," *Network World Fusion*, *http://www.nwfusion.com/reviews/2003/0303patchrev.html*.

Evidence Enhancing Technology

Bridging the Techno-Legal Gap with Secure Audit Logging

by Erin Kenneally

Erin Kenneally is a Forensic Analyst with the Pacific Institute for Computer Security (PICS), San Diego Supercomputer Center. She is a licensed attorney who holds Juris Doctorate and Master of Forensic Sciences degrees.

erin@sdsc.edu

Got Logs?

Computer logs may be used as evidence. Computer logs are like footprints for traditional crime scene investigators or financial ledgers for auditors. All of these objects are time machines, containing answers to questions related to IT processes, unlawful acts, and economic transactions, respectively. Inherent in each is the ability to reconstruct the who, what, when, where, why, and how of an IT, legal, or financial dispute. As crimes and social wrongs increasingly involve or target the use of computers, and as business relies on information systems to function, logs have become the digital eyewitnesses to transactions between computers and humans.

Realizing that eyewitnesses are only as valuable as their perception, memory, and cognition, so too are logs in their ability to paint a picture of digital events. Similarly, just as persons cannot predict or prepare for eyewitness events, it is difficult to foreknow which digital transactions will necessitate recreation in resolving a dispute. However, we can engineer reliable perception, memory, and cognition into our digital eyewitnesses through the process of secure audit logging (SAL).

SAL provides a general model for collecting and storing digital event data in accordance with legal admissibility standards and in compliance with the specific audit needs of systems administrators, law enforcement, and businesses. During legal disputes, investigators – system administrators, forensic examiners, regulators, private and public law enforcement – will often rely on audit and transaction logs as a source of evidence to prove/disprove their claims. These logs can contain virtually any type of data that a computer system is programmed to capture.

The SAL model facilitates the automated, centralized, and trustworthy collection and storage of any audit data that is dictated by a chosen policy. Information assurance and the ability to maintain the integrity of digital data for the purposes of legal proof are continually challenged by the nature of network computing, system bugs and vulnerabilities, and constantly changing technology. These features have conspired to facilitate confusion surrounding the admissibility of log records. The secure audit logging model is being designed with evidentiary standards and presumptions in mind. As such, SAL raises the bar for successful challenges to integrity of log records by including assurances of credibility – authenticity and reliability.

Secure Audit Logging: A Forensics Enhancing Technology

Many of the emerging applications for auditing and investigation are focused on data collection and monitoring within an organization's intranet. Unfortunately, existing tools for facilitating such capabilities require system administrators within these networks, who are not versed in legal principles, to deploy the technology that should enable business process within the context of its policies and legal directives. Such deployments frequently become mired in the difficulties of supporting user functionality, configuring hardware and software for compatibility, and providing other utility services, all within volatile distributed environments. The design purpose for logs produced by computers originated from utility service requirements. For instance, logs were system administrators' way of debugging, or troubleshooting, computers for various technical performance reasons. Absent this data collection mechanism, the ability to detect suspicious behavior and computer intrusions and distinguish between inadvertent machine error and malicious human tampering was implausible. As logs are increasingly being used for purpose of corporate governance, regulatory and legal forces are driving nontechnical folks to turn to log data to substantiate and defend against dispute claims. Because logs are humans' link to the who, what, when, where, and how of computer functioning and usage, logs are being thrust from the annals of computer "techdom" to the adversarial realm of jurisprudence. Like all other evidence offered for legal proof purposes, they must meet certain evidentiary standards.

In general, the standard for the admissibility of evidence is that it is shown to be relevant, authentic, and reliable.

Anatomy of the Law Applied to Computer Logs

LEGAL SEMANTICS AND PURPOSE OF EVIDENTIARY STANDARDS

In general, the standard for the admissibility of evidence is that it is shown to be relevant, authentic, and reliable. This includes that the evidence must not contain hearsay, unless it falls within an exception to the hearsay prohibition. These preliminary determinations can occur under the auspices of the Federal Rule of Evidence requirement that the matter in question is what it is claimed to be, or via the more demanding showing of reliability for scientific, technical, or specialized evidence. The purpose of this initial screening is to ensure that the evidence is reliable enough to go before a fact-finder, whose job it is to decide what weight that evidence should carry in resolving the issue at hand. In other words, a basic evidentiary tenet governing admissibility determinations is that there are guarantees of trustworthiness attached to the evidence so that a jury is not unduly confused or prejudiced.

AUTHENTICATION AND LOG EVIDENCE

Authentication standards are meant to ensure that the evidence is what it purports to be, and how rigorous a foundation is needed to make this finding depends on the existence of something that can be tested in order to prove a relationship between the document and an individual, and control against the perpetration of fraud.

The degree of scrutiny applied to determine whether or not computer log evidence is admissible is unsettled. This determination may turn on how a court categorizes the log evidence: computer-generated, computer-stored, or some hybrid. To date, there is no overarching prescription for establishing how computer logs should be categorized, thus leaving admissibility open to case-by-case determinations.

Generally, the authenticity control is established by testimony that the computer program which generated the record was functioning properly. It is important to keep in mind that this can rebutted if the source, method, or circumstance of preparation indicates lack of trustworthiness.

While increasing automation will diminish the number of witnesses qualified to authenticate computer-generated evidence like logs, inconsistencies at the human-computer interface when collecting, processing, and storing logs may provide fodder for log opponents to rebut the low threshold of proving authenticity and reliability and force proponents of log evidence to offer more solid foundational proof.

SECURE AUDIT LOGS AS THE DIGITAL CHAIN-OF-CUSTODY

The level of scrutiny and legal categorization of computer logs is ambiguous. To be clear, although the legal standard is unwavering – relevant, authentic, original – the application of the standard to log evidence is unsettled. While not comforting for those seeking black letter law on whether logs can be used as a sword or shield in legal disputes, there are controls that courts use to measure the reliability of evidence which can serve as a blueprint for attempts to ensure log admissibility. Arguably one of the most recognized reliability controls is chain-of-custody, and it is this concept that the SAL model mirrors.

Chain-of-custody (COC) is one of the controls used by courts to implement reliability standards. That is to say, authenticity of physical evidence is tested by accounting for the who, what, when, where, and how of a given piece of evidence from its initial discovery, to its collection, access, handling, storage, and eventual presentation at trial. COC has been institutionalized as a procedure for the seizure of physical evidence by law enforcement, as well as for the handling of digital evidence by computer forensic examiners as a measure of evidence integrity. The SAL ensures a digital chain-of-custody so as to minimize the challenges that digital evidence has been created, lost, damaged, or modified. SAL minimizes the manual human interfaces during collection and storage, as well as providing the metrics upon which legal determinations of reliability can be made.

SAL replicates the general procedures followed by computer forensic examiners to establish authenticity of physical evidence. These include:

- refraining from altering the original evidence
- documenting procedures used in collection, storage, and analysis and explaining any changes that may have been made to the evidence. These procedures should be auditable.
- maintaining the continuity of evidence; making a complete copy of data in question using a reliable copy process (independently verifiable; hashing)
- employing security measures (tamperproof storage, write protection)
- properly labeling time, date, source (tracking # and tagging)
- limiting and documenting the persons with access to data

LOG EVIDENCE RELIABILITY CONTROLS – WHAT IS THE LYNCHPIN OF CREDIBILITY?

Is the lynchpin of credibility for log data derived from the technology (computer and software producing the log), or from the person who reads and interprets the log data? In other words, who the real witness is should dictate what should be examined to measure the trustworthiness of statements in the logs. The nature of log evidence, unlike instances where a human is putting a pen to paper, suggests that the "real witness" is the chain of digital events surrounding the creation, transportation, and storage of logs. As such, courts should insist upon controls that measure the reliability with as little abstraction as possible.

Do the controls applied by courts to adjudge reliability log records ensure that evidence standards prescribed by the F.R.E. regulations and policy are being met? Controls are the guarantees of trustworthiness that enable an audit event to be measured against a standard or principle. The value of SAL lies in its ability to provide more direct guarantees of trustworthiness of log records, thereby reducing the uncertainty of legal risks. Even though the reliability controls for paper records are an abstraction of the controls for witness credibility, the underlying metric is the same: time (chronology), distance (location), and computation (cognition). SAL provides controls that more directly measure the lynchpin of credibility – the technology producing the logs – against the relevant standard.

When a witness takes the stand to testify, the audit event is what he is being asked to testify about – i.e., the accident he saw, what he did with the evidence being offered, or whether the computer was functioning – and is manifest by testing the witness's perception, memory, or narration/bias. Audit tools such as oath, personal presence at trial, and cross-examination are used to measure the credibility/trustworthiness of the witness's account of the audit event. The reliability metric the audit tools are measuring can be distilled into time (chronology), distance (location), and cognition.

As logs are increasingly used to resolve legal disputes and become the lynchpin of proof, focus will shift from presumptively ushering in the digital traces of business activities to disputing the logs used to buttress claims. Attempts to discredit logs will accompany this shift, and the technical folks who understand the mutability associated with current log data will be tapped for their knowledge that alterations (insertion, deletion, modification) are not only possible but probable, and oftentimes impossible to detect. This will be exacerbated by the emergence of software programs that expand data alteration capabilities to anyone with point-and-click capabilities, in contrast to the present state of affairs where log data alteration is limited to a small number of persons with the knowledge and skills to manually weave through log data and manipulate certain bits to reflect factual changes. The evidentiary significance is that continued reliance on controls such as proper functioning of the computer producing the logs do not speak to the threats to log integrity. Indeed, one's IDS, spreadsheet program, or email program may be working in tip-top shape, but that does not address the risk that the data it produces was altered by virtue of the interconnections or vulnerabilities posed by other persons and programs.

Two recent cases have turned this conjecture into reality. Log evidence was the subject of scrutiny in the acquittal of a U.K. teen accused of launching a DDoS attack that knocked out IT systems at the Port of Houston in Texas. The striking aspect of this case is that logs were used as both a sword and shield to support the increasingly popular, "unknown third party" defense. On one hand, the defense leveraged server logs showing regular probing of the defendant's computer to assert that it was possible the system could have been compromised and wielded by a remote hacker to perpetrate the crime. Simultaneously, the accused decried that the log files found on his system that implicated him in the attack were unreliable because his system was unpatched and thus susceptible to manipulation.

A similar tactic was used successfully to persuade a jury in a Montgomery County Circuit Court that an accountant charged with tax evasion was not guilty. Here the defendant blamed tax return inaccuracies on an unnamed computer virus. Despite evidence showing that the alleged virus did not affect the tax returns of clients prepared on the same computer, the defendant averted the maximum 33 years in prison and up to $900,000 in penalties.

Whether or not these outcomes are merely exceptional or the tip of an iceberg, they illustrate increasing reliance on digital evidence to fortify a particular rendition of the "truth." As the possibility of backdoors and vulnerabilities in systems challenge litigants to prove a negative in presenting or defending a claim, it becomes all the more

A battle of the logs will demand that the data contained therein is reliable.

important to establish the reliability of the digital footprints that paint the real picture of the "truth." Because logs can be authoritatively persuasive for either party in a dispute, a battle of the logs will demand that the data contained therein is reliable. As such, control mechanisms must be employed to avoid finding reasonable doubt or a preponderance based on logs shrouded by conjecture.

By relying exclusively on humans as the only witness in addressing the reliability of log evidence, courts are not addressing the threats and vulnerabilities attendant to electronic evidence. They overlook the reality that computer hardware, software, and their interconnections converge to produce log evidence that is susceptible to events that render logs unreliable. This is simply an inadequate control to measure reliability. It is similar to claiming that all the cells in one's body can be labeled as trustworthy because of the fact that the body's organs and systems are healthy and functional.

If the law continues to use controls that provide second-order indicia of reliability, business reliance will be the control used to safeguard trust in logs. However, businesses run on commodity technology. The problem with relying on commodity technology to satisfy legal reliability standards is that it is driven by time-to-market forces and not built with legal standards in mind. This is not satisfactory when the costs of mistakes and errors are economically high and socially detrimental. Further, taking judicial notice of a process's accuracy (i.e., that computers produce logs that can be relied on) may be confused with taking notice that a particular result is accurate (i.e., that logs submitted as evidence are trustworthy).

SAL addresses the log challenges by engineering the collection and preservation of logs with the principles and procedures of forensic integrity in mind. SAL offers more empirical evidence of the sequence of events surrounding log collection and storage, as well as minimizing the error that accompanies human interaction with log processes. By performing a digital chain-of-custody , the SAL model better fits the evidence whose trust is attempted to be measured.

The Timing Is Right

The development of this secure audit logging technology is motivated by the need to facilitate a just legal framework for establishing the trustworthiness of digital log records and for recognizing the fundamental uncertainties in the processes involved in utilizing these logs as evidence. These uncertainties are not being addressed by current information assurance and product development processes. IT departments lack meaningful guidance on how to utilize technology to comply with legal/regulatory standards, and reliance on vendors to know and foster the enabling technology is misguided at best. Further, these uncertainties risk being perpetuated if the assumptions underlying legal interpretations of the standards are institutionalized without proper measurement.

This article is an abbreviated version of a much more detailed work in progress. An extended version of this paper, complete with references, can be found at *http://security.sdsc.edu/*.

the bookworm

by Peter H. Salus

Peter H. Salus is a member of the ACM, the Early English Text Society, and the Trollope Society, and is a life member of the American Oriental Society. He owns neither a dog nor a cat.

peter@netpedant.com

Time for the December column again! And I gotta pick my annual holiday "best" list. Let me say here that it was no easy pick. I decided that Gibson, Stephenson, and Waldrop weren't eligible. That's like disallowing Eisenstein, Griffiths, and Kurosawa from a best director's list. Oh, well. How I suffer. (And I'm sure that many of you read *Pattern Recognition* and *Quicksilver* and *Custer's Last Jump* with as much pleasure as I did.)

But there is a consolation: I also get to list a bonus book. Most of the others are large: my bonus will go easily into your shoe (instead of that lump of coal) or into a stocking when furled. It's the latest from Illiad. What? You don't read "User Friendly"? Shocking! Learn how to generate static electricity by rubbing balloons against Dust Puppy.

But I do want to mention a few books before getting to the list.

(In)Security

Bruce Schneier has a machine that enables him to turn out good books with amazing regularity. *Applied Cryptography* (now in a second edition) and *Secrets & Lies* have been reviewed by me previously. Bruce's *Beyond Fear* is, quite simply, in a class by itself. This is not "merely" a book about computer security, it's about just how ineffective those waits in the airport are. It's about why ID cards will be absurd encumbrances and why ID checks are just worthless.

Beyond Fear is a must buy and a must read. If only there were a way to get someone like Tom Ridge or W to read it.

TCP/IP

Bill Fenner and Andrew M. Rugoff have done the networking field a real favor. They have edited and enlarged the first volume of Rich Stevens' standard *UNIX Network Programming*, volume 1. The first edition had 600 pages; the second was just 1000; this one has 946. They've revised what needed to be revised and added great gobs of stuff. And they've deleted where necessary. They've dropped T/TCP and XTI; they added three chapters on SCTP. Etc. It's a great job!

HLA

I'm not a great fan of assembler language, most likely because I've never needed to do a lot of low-level programming. But I can see the utility of it. HLA is "High-Level Assembler" – actually a compiler. Hyde has produced a readable book with a useful CD included.

Peter's Holiday List

This has been a strange year. And I'm breaking what was once a fixed and firm rule: not considering second and later editions. At the same time, there are several reissues I'm most likely slighting. And there are certainly several books of which I think quite highly that I'm not including.

But I do think that the volumes I'm listing are among the very best. For the nth year in a row, let me state that this list is not ordered by rank, nor is it alphabetical. But these are good books. Really good books.

Season's greetings!

Cricket Liu, *DNS and BIND Cookbook* (O'Reilly)

Matt Bishop, *Computer Security: Art and Science* (Addison-Wesley)

William R. Cheswick et al., *Firewalls and Internet Security* (Addison-Wesley)

Eric S. Raymond, *The Art of UNIX Programming*. (Addison-Wesley)

Marcel Gagné, *Moving to Linux* (Addison-Wesley)

Ellen Siever et al., *Linux in a Nutshell* 4th ed. (O'Reilly)

John Eilert, et al., *Linux on the Mainframe* (Prentice Hall)

David Jordan & Craig Russell, *Java Data Objects* (O'Reilly)

Bruce Schneier, *Beyond Fear* (Capricorn Books)

W. Richard Stevens et al. *Unix Network Programming, Volume 1: The Sockets Networking API*, 3rd ed. (Addison-Wesley)

Bonus Book:
J.D. Frazer, *Even Grues Get Full* (O'Reilly)

BOOKS REVIEWED IN THIS COLUMN

BEYOND FEAR
BRUCE SCHNEIER
New York: Capricorn Books, 2003. Pp. 295. ISBN 0-387-02620-7.

UNIX NETWORK PROGRAMMING, VOLUME 1: THE SOCKETS NETWORKING API
W. RICHARD STEVENS ET AL.
3rd ed. Boston, MA: Addison-Wesley, 2003. Pp. 946. ISBN 0-13-141155-1.

THE ART OF ASSEMBLY LANGUAGE
RANDALL HYDE
San Francisco, CA: No Starch, 2003. Pp. 903 + CD-ROM. ISBN 1-886411-97-2.

Nearly 20 Years Ago in U[SE]NIX

by Peter H. Salus

USENIX Historian

peter@netpedant.com

At the beginning of 2002, I reminisced about the January 1987 USENIX Conference in Washington, DC. The second Washington snowstorm . . .

Ted Dolotta, a contributor to the -mm macros and one of the managers of USG, sent me these remembrances:

"The reason for this message is your "Fifteen Years Ago in USENIX" item in the February 2002 issue of *;login:* – in it you mention (in addition to the 1987 USENIX meeting) the 1984 USENIX meeting in DC, snowstorm and all. That started me reminiscing.

If my memory serves, the January 1984 USENIX meeting was also marked by another "storm:" there was a last-minute, surprise exhibitor at that show: Big Blue, which set up a dozen PC-ATs in a hotel suite (all regular exhibit space being already taken) running the Personal Computer Interactive Executive, PC/IX, a single-user UNIX running on the PC-AT and developed for IBM by my team at INTERACTIVE Systems. (I just looked at the User's Manual for PC/IX, and it says, smack in the middle of the title page, "by INTERACTIVE Systems Corporation," with the IBM logo relegated to the bottom of the page.)

IBM invited all the attendees to come up to their hotel suite and play with the system at will; there were no canned demos, no presentations – just UNIX and a bunch of IBM guys, and my folks, to answer questions. (Several of my guys had to buy, on short notice, their first adult suit; two among them actually asked me whether they could share a suit.)

And notwithstanding the fact that PC/IX eventually went nowhere, suddenly UNIX was no longer a Bell Labs/Berkeley/academia/hacker/nerdy thing – it was in the mainstream, endorsed by the largest computer company in the world. A heady day, indeed! The whole thing was just amazing (I know I'm biased).

As I said, PC/IX was not a commercial success; it was followed by VM/IX (UNIX as a guest on the VM/360 mainframe system) and IX/360 (native UNIX on a System/360 mainframe); both of these flopped as well. And then came AIX: UNIX on the PC-RT (a RISC chip), which IBM sells to date, albeit on much more modern hardware. These ports were done by my team at INTERACTIVE (another outfit whose name escapes me started the IX/360 port, but eventually INTERACTIVE was asked to finish it in collaboration with IBM/Germany). For AIX, that team consisted of 18 people, including the support staff – secretary, hardware guy, administrator, etc.; IBM had a team of 350+ people in Austin testing the stuff my folks built. IBM was very worried about keeping the project secret: we were not allowed to open the window shades in our offices, and the PC-RTs were chained to the walls. I did tell you about the AIX manual and their problems in a previous letter.

Speaking of manuals, I also explained in another letter about the various issues that arose in the context of creating UNIX manuals within the constraints of IBM's practices – it was essentially Mission Impossible; but to give the devil his due, IBM graphic design folks did a *great* job of designing the covers for the PC/IX and VM/IX documentation (whose content was *identical* except for the name of the system and for some SysAdmin stuff, since PC/IX was a *single-user*, native-mode system that ran on the PC-AT, while VM/IX was a *multi-user* system hosted on VM/360 – something I'm quite proud of to this day). Anyway, the PC/IX binders were pin-striped, very dark charcoal gray, with white type, and a bud vase with a *single* red rose, harking back to the original IBM PC ad campaign featuring "The Little Tramp" (Charlie Chaplin look-alike with a red rose); the VM/IX binders were *identical*, except for a vase with a *bouquet* of red roses. It was brilliant.

Today, IBM is into UNIX in a big way, with Linux mainframes and AIX systems (the latter, I suspect, will be around alongside Linux systems for a good long time), huge booths at Linux World, and a suit from SCO over alleged license and copyright infringements . . .

But it was at the 1984 USENIX meeting in Washington, DC, that IBM first publicly put its toe into the UNIX stream (no pun intended)."

Thanks, Ted. And a happy (appropriate) holiday to all of you. One wonders whether SCO realizes that INTERACTIVE Systems wrote the first UNIX port for IBM . . .

2004 marks a number of things: The ARPAnet/Internet and UNIX will both be 35; and Linux 1.0 will be ten. Lots of fodder for a historian.

I hope to celebrate these events live at Nordu in Copenhagen in January, at USENIX in Boston in July, and at SANE in Amsterdam in September.

USENIX news

USENIX MEMBER BENEFITS

As a member of the USENIX Association, you receive the following benefits:

FREE SUBSCRIPTION TO *;login:*, the Association's magazine, published six times a year, featuring technical articles, system administration articles, tips and techniques, practical columns on such topics as security, Tcl, Perl, Java, and operating systems, book reviews, and summaries of sessions at USENIX conferences.

ACCESS TO *;login:* online from October 1997 to last month <*www.usenix.org/publications/login/login.html*>.

ACCESS TO PAPERS from the USENIX Conferences online starting with 1993 <*www.usenix.org/publications/library/index.html*>.

THE RIGHT TO VOTE on matters affecting the Association, its bylaws, election of its directors and officers.

DISCOUNTS on registration fees for all USENIX conferences.

DISCOUNTS on the purchase of proceedings and CD-ROMS from USENIX conferences.

SPECIAL DISCOUNTS on a variety of products, books, software, and periodicals. See <*http://www.usenix.org/membership/specialdisc.html*> for details.

FOR MORE INFORMATION REGARDING MEMBERSHIP OR BENEFITS, PLEASE SEE <*http://www.usenix.org/membership/*

OR CONTACT

<*office@usenix.org*>
Phone: 510 528 8649

2004 Election for Board of Directors

by Ellie Young

Executive Director

ellie@usenix.org

The biennial election for officers and directors of the Association will be held in the spring of 2004. A report from the Nominating Committee will be emailed to USENIX members and posted to the USENIX Web site in mid-December 2003 and will be published in the February 2004 issue of *;login:*.

Nominations from the membership are open until January 9, 2004. To nominate an individual, send a written statement of nomination signed by at least five (5) members in good standing (or five separate nominations), to the Executive Director at the Association office, to be received by noon PST, January 9, 2004. Please include a Candidate's Statement and photograph to be included in the ballots.

Ballots will be sent to all paid-up members on or about February 6. Ballots must be received in the USENIX office by March 19, 2004. The results of the election will be announced on the USENIX Web site by April 2 and will be published in the June issue of *;login:*.

The Board consists of eight directors, four of whom are "at large." The others are the president, vice president, secretary, and treasurer. The balloting is preferential: those candidates with the largest numbers of votes are elected. Ties in elections for directors shall result in run-off elections, the results of which shall be determined by a majority of the votes cast. Newly elected directors will take office at the conclusion of the first regularly scheduled meeting following the election, or on July 1, 2004, whichever comes earlier.

USENIX SUPPORTING MEMBERS

Ajava Systems
Aptitude Corporation
Atos Origin B.V.
Computer Measurement Group
Electronic Frontier Foundation
Interhack Corporation
MacConnection
The Measurement Factory
Microsoft Research
Motorola Australia Software Centre
Sun Microsystems, Inc.
UUNET Technologies, Inc.
Veritas Software

conference reports

OUR THANKS TO THE SUMMARIZERS:

Scott A. Crosby
Catherine Dodge
Clif Flynt
Seung Won Jun
David Molnar
Chris Ries
Gelareh Taban
Tara Whalen

NOTE

Reports for BSDCon '03 arrived too late to be included in this issue. They are available at *http://www.usenix.org/publications/library/proceedings/bsdcon03/confrpts.*

12th USENIX Security Symposium

August 4–8, 2003 Washington, D.C.

KEYNOTE ADDRESS: REFLECTIONS ON A DECADE OF PSEUDONYMITY

Black Unicorn

Summarized by Tara Whalen

Black Unicorn, aka A.S.L. von Bernhardi, kicked off the conference with his talk on the relationships between identity, reputation, and trust. Anonymity has negative connotations, or "sunny climes attract shady characters." He chose his pseudonym when the cypherpunks list was being archived, and suggested that anybody who doesn't want to attract undue attention should probably pick a benign pseudonym, such as "Bill Smith."

Black Unicorn and Vern Paxson

He then launched into a detailed discussion of identity: its definition, its value, and some common fallacies. The definition he prefers is "the distinct personality of an individual regarded as a persisting entity; individuality."

The value of identity encompasses both its use as a unique identifier and in establishing the uniqueness of a reputational assertion.

The problems with identity are a lack of a standard unique identifier, reliance on third-party attestation, and the absence of static physical characteristics for identification. In addition, the link between identity and reputation is not addressed in conventional systems.

This led to the next phase of the talk: reputation. Black Unicorn's most appropriate definition is a specific characteristic or trait ascribed to a person or thing: a reputation for courtesy. The value of reputation is as a predictor of behavior, as a means of valuation, and as a means for third-party assessment. Black Unicorn took issue with the notion of reputation as a behavior-predictor.

Black Unicorn then discussed trust; he presented several definitions, choosing "reliance on something in the future; hope" as the most appropriate for this talk. The value of trust has several components: as a means of inexpensive due diligence; as a delegation enabler; as a means to reduce robustness of environmental security; and as an indication of risk tolerance.

Black Unicorn then discussed the relationships between trust, identity, and reputation, showing how factors such as credentials, third-party attestation, and certification affect trust. Identity information can be augmented by due diligence, consequence augmentation (e.g., penalties for crimes), and third-party attestation.

Q: Given the reticence of Americans to use ID cards, how will we be able to identify people?

A: This problem is likely to remain. There is no good answer; identity is always going to be nebulous. We have different roles that we play, and we tend not to like third-party assertions about our identities.

Q: How is the use of third-party attestations different from due diligence?

A: We often look at collections of attestations, which is better than looking at only one attestation. The problem with third-party attestations is that these parties have limited motivation to do proper due diligence (to save money), and their own liability is limited by insurance, which makes them less likely to be appropriately diligent. Direct experience and/or lore may be better guides.

Q: Do you want different ID documents for different purposes?

A: Definitely. You have to look at the agency making the attestation—what are they qualified to say? Probably not a lot. When looking at ID documents, you have to look at what the agency is good at attesting.

Q: Considering this issue from a cultural perspective: Identity is partially defined by race, religion, etc., and this affects one's reputation. Comment?

A: I agree. A lot of reputation is tied up in name, for example, and differs from culture to culture (e.g., Europe vs. US). Look at what's used for credibility: in Europe, it's family history, unlike in the US.

REFEREED PAPERS: ATTACKS

Summarized by Clif Flynt

Remote Timing Attacks Are Practical

David Brumley and Dan Boneh, Stanford University

The authors received the Best Paper award for their report on the viability of timing attacks on the RSA algorithm as implemented in OpenSSL. They proved that they could extract RSA private keys, not only when running on the same hardware as the secure application, but also across a network including several routers.

Dan Boneh described how a timing attack works and how to defend against it. In a timing attack, the secure server is sent many different queries that are expected to fail, and the time it takes to reject each query is measured. Rejecting a query requires that the server perform several math operations. Depending on the organization of the 1s and 0s in the query, different speed optimizations may be performed. A program can deduce the positions of 1s and 0s from the time required to perform the math operations.

The timing attacks have been done many times in the past, but previous work was done against "simple" devices like smartcards. Boneh and Brumley's work shows that a surprising level of noise can be introduced into the timing data without degrading it beyond usability. Complex systems with multiple applications running simultaneously, or even the variance of network latency, do not produce enough noise to disguise the time difference required to perform the analysis.

OpenSSL was chosen for this work because the package is used in many other applications, including mod_SSL, stunnel, sNFS, and more. Of the applications examined, only the Mozilla NSS packages defaulted to secure behavior.

Because the RSA algorithm requires many multiplications of very large numbers, most implementations use arithmetic optimization techniques to speed up encrypting and decrypting. These techniques are sensitive to certain bit patterns in the test and real key, making a message rejection faster or slower depending on how closely the bit patterns of the test key match the real key. A defense against this attack is to use RSA blinding, in which the client and server should decrypt a random string as well as the actual encrypted text to provide a random decryption time. This produces a 2–10% hit in performance. Brumley found that this was sufficient to prevent a successful timing analysis. This is implemented by default in versions of OpenSSL after version 0.9.7b.

802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions

John Bellardo and Stefan Savage, University of California, San Diego

While many members of the audience were connected to the outside world via their laptops and 802.11b wireless links, Stefan Savage demonstrated just how insubstantial that link can be. He opened his talk by showing a graph of current network activity and then attacked John Bellardo's link, halting his download. When this attack was started, the graph showed that traffic to Bellardo's laptop was reduced to nearly 0 packets, and Bellardo concurred that he was no longer downloading any data. Savage then extended the attack to the rest of the audience, and I can confirm that my connection to the outside world was gone. This lasted a few seconds after which Savage discontinued the attack, and service was restored.

He then described the two denial-of-service attacks he used, both of which use legitimate features of the 802.11b protocol. The first attack, which disabled only a single 802.11b user, used a deauthentication attack. In a normal conversation between an 802.11b client and access point, the client requests and receives an authentication. At any point in the conversation, the client can request that the session be deauthenticated. The deauthentication request is not a secure message; thus one client can send a message to deauthenticate another client. Once a session has been deauthenticated, the original client must return to the beginning of the conversation with the AP before transmitting any more information.

He described a simple technique to protect against this attack. When the AP receives a deauthenticate request, it should hold the request for a period of time before implementing it. If a fresh data packet from the deauthenticating client is received, the AP will not process the deauthenticate request.

Savage described other potential attacks based on fields designed to enable power conservation and reduce packet collisions. However, when they implemented some of these attacks they discovered that current implementations of most 802.11b interface cards don't implement these features, and data in those fields is ignored.

As the use of 802.11b becomes more widespread and quality-of-service demands become greater, we can expect new cards to implement these features, so Savage simulated a network of cards

A book-signing party

that implement the specifications correctly, and simulated attacks on that network. One of these specifications is the Network Allocation Vector (NAV) field. The NAV field allows an AP-client pair to reserve a period of time for their exclusive use. This allows a large packet to use up the bandwidth without having another unit collide with it. This time period can be as long as 31.5 milliseconds. The NAV bit is maintained by the firmware and, in theory, can't be modified by a user program. Savage described how an attacking application can scan the SRAM to find the value being placed into the packets and modify it. The defense against this attack is simply to reject unreasonably large values in the NAV field. In actual use, legitimate values are always under 3 ms.

Denial of Service via Algorithmic Complexity Attacks

Scott A. Crosby and Dan S. Wallach, Rice University

Scott Crosby pointed out that algorithms have best-case, normal-case, and worst-case behaviors. When we implement algorithms, we rely on normal-case behavior. A system with malicious users may be forced to demonstrate worst-case behavior.

A hash algorithm displays a constant time to access an element (O(1)), unless there are many collisions. As the number of collisions increases, the behavior of the hash access approaches linear (O(*n*)). By discovering a few sets of letters that hash to "zero," an attacker can easily generate a large number of collisions.

Crosby demonstrated that thousands of collisions are required before the linear nature of the hashtable search becomes
a problem, but that this can be achieved in six minutes with a typical dialup modem. In an attack on a Perl application, he generated 90,000 collisions. After these collisions, retrieving a value for a key with no collision took about two seconds, while retrieving a value for a key that had collisions took about two hours.

This vulnerability was found in Perl, Squid, the Bro Intrusion Detection System, the Linux routing cache and directory entry cache, and others. Since the paper was written, the Bro IDS and the Linux vulnerabilities have been addressed.

Crosby described several alternative hash implementations and discussed their strengths and weaknesses. The universal hashing algorithm described by Carter & Wegman in 1979 is about as fast as the Perl 5 hash algorithm. The UMAC hash generator is optimized for modern compilers. A new hashing library dubbed UHASH was developed based on the Carter & Wegman code and UMAC code with further optimizations and generalizations to make it useful for applications with unknown length strings (CGI applications that may hash on user input), as well as applications where a key length can be determined (compilers where the keyword size is fixed).

The UHASH library is slower than the Perl hash algorithm for strings of fewer than 60 characters, but faster for longer strings.

INVITED TALK: DISTRIBUTING SECURITY: DEFENDING WEB SITES WITH 13,000 SERVERS

Andy Ellis, Akamai

Summarized by Seung Won Jun

Andy Ellis showed the audience some numbers to give an idea of the scale of the Akamai network: 14,000 servers that run 100 applications and are distributed in 2,400 different locations. They collectively serve 25 billion connections and 200 terabytes per day. Securing such a system is a challenge.

A traditional Web service model may provide confidentiality and integrity via the separation of application/database servers guarded by firewalls and IDSes, but it does not provide availability very well. Availability can be compromised when flash crowds or denial-of-service attacks occur. Akamai addresses it by delivering contents from the edge. While some servers on the edge may be overwhelmed, plenty of others can still serve the requests. The principle of delivering from the edge applies to almost all protocols Akamai supports, including DNS traffic as well as Web traffic.

Several management techniques were presented. Given so many servers and so few administrators (about 30), installing software could be a headache without an automated process, which is called "net deploy." Automated installation gives flexibility for managing servers. If a suspicious event occurs, the relevant servers can be wiped out and freshly reinstalled rather than examined and patched. Key management is done cooperatively by several components: key generation center, key distribution center, access control database, and audit server. An AuthGate is a gateway for access management. Administrators ssh to the AuthGate, which is replicated, to access edge servers rather than having direct access to them. This way, access control policy can be more flexible. For example, the policy can say that a particular user is allowed to access a certain number of machines rather than having the list of machines that are allowed to a user. Edge Diagnostics is the facility to test edge servers. The primary purpose is, hopefully, to show customers that the problem is not in Akamai's servers but somewhere else (maybe in the network). With so many servers, it is important to know what happens to which servers. Event management also requires a scalable solution. Events from several edge servers are merged into a log file, which is collected by Query Aggregator. The Monocle application automatically checks the Query Aggregator and reports any alerts to clients.

While Akamai runs many servers, it does not own, or have control over, any network backbone. Ellis mentioned that BGP is not particularly good about performance or reliability; it is all about "screwing the neighbors" or "not having your bits on my network." Though Akamai is across the street from MIT, Internet traffic requires 15 hops and is routed through New York. On September 11, this became 54 hops, with traffic routed through Israel. To mitigate BGP's routing, Akamai uses some servers to relay the traffic, which is especially effective for transcontinental traffic, producing up to 40% improvement in round-trip time.

Ellis concluded the talk by giving three lessons he had learned: plan for failure, use heavy automation, and make a decision in advance.

REFEREED PAPERS: COPING WITH THE REAL WORLD

Summarized by Clif Flynt

Plug-and-Play PKI: A PKI Your Mother Can Use

Peter Gutmann, Auckland University

In contrast to the many very technical papers regarding timing attacks, encryption, file systems, and packet protocols, Peter Gutmann examined the user experience of public key infrastructure (PKI) to obtain a certificate, and proposed some solutions.

He noted that using the current public key infrastructure is more difficult than it should be. While obtaining a connection to an ISP can be done in a few minutes with three pieces of information (login name, password, and credit card number), obtaining a certificate from a public certificate authority (CA) can take a skilled user between 30 minutes and a month, and may or may not have any verification.

In practice, most sites use the sample (clown suit) certificate that can be generated for testing purposes with OpenSSL, cryptlib, and others. What is needed is a system with the following prime directives:

- Don't scare the user. The certificate request forms should only require information they will actually use (username, password), not unnecessary information such as passport number.
- It must be possible to bootstrap the procedure with no previous certificate. Username/password should be all that's required to get a basic certificate.
- The user can submit the certificate to the CA to get it signed. The CA will return a signed certificate to the user.

The system Gutmann developed relies on a few assumptions: (1) The user has some existing relationship with the certificate-issuing agency. He proposes that banks offer CA services, since a user can be assumed to have a relationship with a bank that has already been verified. (2) There is a centralized server that can act as a CA locator. This is much like a DHCP server. In practice the HTTP 3xx Redirect message was used to redirect a user from a central site to the CA. (3) The resulting procedure need only be as secure as the applications warrant. This is for online shopping, not exchanging nuclear missile launch codes.

The first problem in obtaining a certificate is finding a certificate authority. This can be implemented by having ISPs provide a redirect from a common named location—for example, *http://pkiboot.example.com* or *http://www.example.com/pkiboot*, and using the HTTP redirect to redirect a browser (or automated certificate acquisition program) to the proper page. Current PKI systems use a baby duck security model. The user imprints on the first available system, and believes it to be secure. When started, an automated certificate-acquisition system will be initialized to remove existing state. It will trust the first certificate authority it discovers.

This is obviously insecure, since it is based on physical location. While there are many PKI RFCs in existence, none of them met Gutmann's requirements, so he came up with a partially home-grown solution. A user simply enters his or her username and password and obtains a certificate. A developer can create a Plug-and-Play PKI session that performs PKIBoot using username + password, generates signing key, requests signing certificate using username_password, and generates encryption key. This can be used with files or smartcards for key storage. Gutmann commented that SCEP (Simple Cert Enrollment Protocol), which is used in IPSec routers, is somewhat quirky. The certificate messaging is done with secured messages, but certificate fetch is done via (insecure) HTTP GET. The initial bootstrapping procedure is difficult, and there is no provision of rMAC'd messages. The SCEP system also uses all-in-one certificates, with no separation of signing and encryption.

Analyzing Integrity Protection in the SELinux Example Policy

Trent Jaeger, Reiner Sailer, and Xiaolan Zhang, IBM Research

The SELinux project is an attempt to provide mandatory access controls to Linux. The project includes a set of example rules that are intended to provide a secure base for developing local rule sets. Jaeger described a technique for analyzing those rules to determine whether or not the example policy is actually secure. This technique relies upon the Gokyo tool, which compares a set of SELinux policy rules and the desired integrity goals and reports how well the rules implement these goals. This work is done as part of the Linux Analysis Tools project, housed at *http://www.research.ibm.com/vali.*

The Linux Security Modules project provides a framework for implementing a set of Mandatory Access Control (MAC) rules within the Linux kernel. The SELinux project is developing a set of rules to be implemented by the LSM framework to implement a comprehensive integrity policy. For each application there are policy statements which define a particular threat model and the reaction to these threats. This can lead to many statements for each application on a Linux system. The SELinux policy base is composed of over 50,000 policy statements, making manual coverage analysis difficult.

At a system level, the SELinux defines an aggregate of the application policies. There is no coherent threat model, and the application policy interactions are not examined. This leads analysts to express concerns regarding the complexity and size of the system being analyzed.

Jaeger and his associates claim that the complexity is necessary because it flows from flexibility in the system. They also point out that while the policy base is large, they can reduce size. Only a smaller policy subset is needed to express an adequate Trusted Computing Base (TCB) rule set. Their technique is to assess the threats and evaluate the policy against TCB security requirements in the threat model. The goal of a Trusted Computing Base is to protect higher integrity data and applications from modification or misbehavior by lower integrity applications.

Security Holes . . . Who Cares?

Eric Rescorla, RTFM, Inc.

Eric Rescorla reported on research into user compliance with security upgrade notifications. They probed many machines to see what versions of OpenSSL were installed. They discovered that most sites that will install an upgrade (about 40%) do so immediately when an update is released. The next surge in updates (about 20% of sites) happens when an exploit is released. The number of patched sites asymptotically approaches 35%. They found that large installations are more responsive than small sites. This can be explained by large installations having a full-time administration staff responsible for installing upgrades.

During the question period, one person involved in writing up descriptions of security flaws wasn't aware that disabling SSLv2 would provide a workaround. A reward/punishment system for installing or not installing security upgrades might work, but nobody knows where the money for rewards would come from. It was observed that the OpenSSL upgrade was not trivial; if upgrades are not easy, they won't be installed. Rescorla was asked what percentage of sites surveyed were home-user systems. He replied that this was not known, and the number of poorly administered home systems is a serious problem.

INVITED TALK: PROTECTING THE INTERNET INFRASTRUCTURE

John Ioannidis, AT&T Labs–Research

Summarized by Seung Won Jun

The Internet has infrastructure, something upon which applications depend: links, routers, supporting services such as DNS servers and perhaps Google, and buildings where the equipment is located. Before getting into details, John Ioannidis mentioned several security mantras. There is no global security solution, so we must remain vigilant. Since security is always a cost-benefit trade-off, it is important to understand the threat model (who is out to get us and how they might get us), trust model (who are friends and from whom can we get help), and available tools (or, in the end, money, which is not sufficient but is necessary for security).

Infrastructure consists of several layers. Physical infrastructure includes fiber, wires, routers, buildings, and electric power. To protect the physical infrastructure against intruders, natural and manmade disasters, and other accidents, we can use hardening and replicating in conjunction with traditional measures such as alarms, locks, traps, and armed guards.

Bit transport is the infrastructure in which bits travel. Links are characterized by capacity, delay, and bit-error rate. Attacks are typically on capacity, that is, denial of service. According to a four-year-old survey of attacks, the DoS attacks are fairly crude and anisotropic, which means that attacks are directional rather than pervasive, probably because attackers can take control of only a limited number of hosts. What can we do about the DoS attack? We can detect the attack by monitoring traffic carefully or even marking some traffic. Upon detection, although we may not completely stop the attack traffic, we can reduce its collateral damage. A "blackhole" router, which is configured manually or semi-automatically, swallows the attack traffic rather than forwarding it to the victim. As a result, other parts of the network are saved from the attack. Going even further, we can "push back" the attack traffic by making a router tell its upstream routers not to forward the attack packets.

A major component of control infrastructure is DNS. There are many points where DNS can go wrong. While the root and TLD DNS servers are critical, there are so few of them that they can be, and are, DoS-attacked. The DNS response, which is carried on UDP, is not authenticated and, hence, can be spoofed. Cache poisoning (corruption of local DNS servers) can misdirect the traffic, and the servers are prone to be misconfigured. Although DNSSec tries to address some security aspects in DNS, it still leaves many problems unsolved. A semantic problem is fundamental: Does microsoft.com represent "the" Microsoft that you intend? Availability is still not addressed, and the key management for DNSSec will be a nightmare. The configuration of DNSSec is more difficult than DNS, and we have little operational experience.

Another component of control infrastructure is routing, particularly focused on BGP. As BGP is arguably the most distributed routing protocol, its complication is aggravated by the policy conflicts among ISPs. Route announcements are not authenticated, which can lead to a problem. Considering the effectiveness of such a simple measure as filtering out all BGP packets whose TTL is lower than 254, it is a pity that not all BGP routers follow such a practice. S-BGP, So-BGP, and IRV address the security issue of BGP, but security is always harder to bolt on later than to build in from the beginning.

PANEL: ELECTRONIC VOTING SECURITY

Summarized by Scott A. Crosby

The panel for electronic voting security included Dan Wallach as moderator, Jim Adler from VoteHere, David Dill from Stanford, David Elliot from Washington State's Office of Secretary of State, Douglas W. Jones from University of Iowa, Sanford Morganstein from Populex, and Aviel Rubin from Johns Hopkins University.

The panel started off with a warning by Dan stating that "no blood will be spilled," and asked if anyone from Diebold was in the audience. This was timely because two weeks before the conference, a highly critical report on Diebold voting machines was released. The authors of that report included Avi Rubin, the first speaker, and Dan Wallach, the moderator.

Aviel Rubin started with highlights from the report and a rebuttal from Diebold. He discussed high-profile disclosures and the risks of these disclosures: legal action against the researchers, restraining orders against publishing, PR blitzes to discredit the researchers, and jeopardizing one's job or career. He also made some firsthand observations of the response he experienced: people criticizing the paper without reading it, people calling his boss to complain, and detractors lying to the press and playing on emotion.

The next speaker was Doug Jones, a computer scientist who also sits on the Iowa Board of Examiners for voting machines. As part of that duty, he has assessed voting machines presented by various vendors, including Global Election Systems (later bought by Diebold). He said that in 1997 he talked to the head developer at GES about its flawed use of a static constant as an encryption key, a flaw that still existed over five years later in the code examined by Avi Rubin. He spent most of his presentation documenting that claim and how it shows that the voting machine certification system is demonstrably flawed.

After Doug Jones came Jim Adler. VoteHere does not make voting machines, but it creates software technology for

L. to R.: Aviel Rubin, Dan Wallach, David Elliott, Jim Adler, Douglas W. Jones, Sanford Morganstein

machines. Doug classified attackers as insiders and outsiders and pointed out the need for threat analysis and open technology. He described his company's voting system which uses cryptographic magic to construct ballots and secure shuffles to preserve anonymity. He emphasized the importance of looking at requirements, design, and risk analysis.

David Elliot described the problems of the current system as the result of benign neglect. It's designed with security based on controlled access to the system. He noted that there were no security standards until 1990, and even those are voluntary. The current testing methodology is hardware stress tests like heat and vibration, not security.

Next came Sanford Morganstein, a representative from Populex, a company that actually manufactures voting machines and is brand new to the field. Their system is a reimplementation based on the Mercuri Method, in which a computer voting machine prints out a human-readable receipt with a bar code.

The final panelist was David Dill. He focused on the cultural gap between computer scientists and voting officials. Voting officials believe that black-box testing can detect malicious code and refuse to listen to computer scientists who say differently. Short-term problems required fixing the regulatory and certification framework and stopping the acquisition of paperless voting machines. He described the long-term problem as satisfying the requirements that lead to touch-screen voting, such as a dislike of paper.

The panel then opened to questions. One questioner inquired whether open source is necessary for secure design. The response from Jim Adler was that all that is needed is verifiability of results. A point was made that even if open source was used, no one could know that a particular program was what was actually being run in the machine. There was disagreement between panelists over the need for a paper trail. Jim Adler was critical of a proposed bill that required one, claiming that it would be stupid and proscriptive; his design uses cryptography that would make paper obsolete. Aviel Rubin disagreed; a mechanism only Ph.D.s understand will be less trusted by the general public than paper.

INVITED TALKS: INTERNET SECURITY: AN OPTIMIST GROPES FOR HOPE

Bill Cheswick, Lumeta

Summarized by Tara Whalen

Bill Cheswick provided a historical perspective on computer security, explaining why he remains optimistic that good security is possible (despite his statement that "an optimistic security person may be an anti-job requirement"). Back in 1993, when he and Steve Bellovin were writing the first edition of *Firewalls and Internet Security*, the Web had not yet arrived on the scene and most network attacks were theoretical. There was no wide-scale sniffing, no massive denial of service attacks, not many worms. But fast-forward only a few years, and these attacks have become prevalent, coupled with a rapid rise in the use of the Internet. Cheswick stated that "there are lot more players, and on average they are a lot less secure."

However, there are also a lot of tools available that weren't around in 1994, such as widely available crypto, SSH, firewalls, and intrusion detection systems. Many of these can be easily deployed, as you don't have to "roll your own" security tools anymore.

There are a number of reasons why Cheswick remains optimistic: Reliable systems can be built from unreliable parts; we have control over the rules we set on our hosts; good encryption is readily available; and "the Bad Guys are giving us lots of practice."

Cheswick pointed out that a cost vs. benefit analysis must be performed: We need to figure out the value of our assets, and how much an attacker is willing to spend. Security is not perfect but only needs to be "good enough." There are some problems that resist easy solutions, such as buggy software, poor password choices, and social engineering. Also, even experts can't always get things right. To illustrate this point, Cheswick displayed some passwords that had been sniffed during the conference from the USENIX wireless network.

To mitigate these problems, he proposed several security strategies. First, stay out of the game if possible ("best block is not be there"), through such means as avoiding the monoculture of homogeneous systems. Second, deploy defense in depth by engineering redundancies into your systems. Third, make security as simple as possible: Set up secure defaults and use hardware tokens. Finally, design security into a system from the start, because it can't simply be added later.

Cheswick continued his talk with a discussion on firewalls. Although they are useful in many situations, they have certain drawbacks. For example, people go around them, and they offer no protection from insiders. Another problem is that firewalls are often used as perimeter defense around very large perimeters; Cheswick believes that smaller enclaves are much safer. Note that you don't have to use a firewall. Cheswick stated that this is like "skinny-dipping on the Internet: somewhat exciting, but with an element of danger." Such an approach requires secure host technology, like the current efforts in *BSD and Linux. One technique is to jail servers (and clients), for example, through chroot. Cheswick provided a list of "routes to root" that should be minimized (such as root network services and setuid programs), and stated that chroot is the only standardized layer of defense that we currently have. He described his experiences with jailing programs, outlined some practical difficulties of this approach, and listed some of the programs he has jailed (Web servers, Samba) and those that probably should be jailed (Apache, NTP).

Cheswick next provided an interesting diversion about "spook networks," telling the audience to talk to spooks for their advice (rather than their secrets). He said that they seem to have a great deal of success running secure networks, and have adopted good practices for maintaining secure systems (e.g., using enclaves and restricting client software). Cheswick finished his talk with a security wish list, which included more work on chroot, formal analysis of crypto, and sandboxes for browsers. He concluded with his contention that things can get better, with enough work and diligence. The audience responded to this talk with anecdotes and opinions, as well as a few questions:

Q: What's the worst thing you could do to the Internet?

A: I don't want to give specifics, but the worst thing I can think of would take it down for weeks. As in the past, I expect that experts would step in to respond immediately, but they'd probably all have to phone each other.

Q: You didn't mention how to help users operate security tools correctly.

A: I don't think that users are going to become more competent. My hand-waving answer is to use tools like USB dongles, obtained from a trusted source.

REFEREED PAPERS: HARDENING I

Summarized by Chris Ries

PointGuard: Protecting Pointers from Buffer Overflow Vulnerabilities

Crispin Cowan, Steve Beattie, John Johansen, and Perry Wagle, WireX Communications

The goal of most attackers in exploiting vulnerabilities is to execute code that they provide, commonly known as shellcode. Usually this is done by overwriting a pointer and aiming it at their own code that has been placed somewhere within the attack space. Different approaches can be taken to protect these pointers, such as surrounding them by canary values that expose an attack if they are overwritten. The authors, however, take the approach that pointers are dangerous until they are loaded into a register, and so their method involves encrypting the pointer until this occurs. During an attack, when the pointer is de-referenced after being overwritten, it will not jump to where it was intended, since the value it was overwritten with will be decrypted. Instead, it points off into some "crazy space" and the program crashes. PointGuard works at compile time and essentially XORs pointers stored in memory with a secret key. This key is kept on its own page, and the page is marked read-only so that the attacker cannot overwrite it. It can be implemented at different times during the compilation process – either at the pre-processor stage, intermediate representation, or the architecture-dependent stage – but the authors decided to implement at the intermediate level. This stage is late enough to avoid most chance of the defenses being optimized away, and early enough to still have type information to distinguish between pointer and non-pointer data. There are some difficult issues that arise while using PointGuard, such as mixing PointGuard-compiled code with code that was compiled without it. For this, the authors added compiler directives. Another major problem, preventing cleartext leaks when register values are stored on the stack to free them up for other purposes, will be prevented in future implementations.

The authors were surprised to discover that in some situations code compiled with PointGuard actually performed better than code that was not. This is probably because PointGuard uses registers more heavily than the normal compiler. Other performance costs varied from less than 1% to 21% overhead.

Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits

Sandeep Bhatkar, Daniel C. DuVarney, and R. Sekar, Stony Brook University

Using a language such as C or C++ allows the programmer to have greater control over memory with tools such as pointers, but this opens programs up to memory errors such as buffer overflow vulnerabilities. One possible solution is to use a type-safe language, but C is too widely used today to be completely abandoned. Other solutions involve educating the programmer or ensuring that the programmer's assumptions about input are always valid. Instead, the authors provide a solution that involves directly stopping the attacker. In order to exploit these memory errors to gain control of the program, the attackers need to know such data as the distance between the buffer their input is placed into and the return address or the memory address of the buffer itself. By randomizing the virtual addresses of the memory the program uses, the attacker will jump to a random location and crash the program most of the time, which provides a telltale sign of the attack. Similar solutions are being used in both the PaX project and PointGuard.

Address obfuscation is performed during linking, loading, and execution. The authors' method essentially involves three different ways to obfuscate: (1) Randomize the base address of the stack, heap, and code memory regions, as well as the starting address of dynamically linked libraries; (2) add random gaps within the stack between allocated memory on the heap, and to routines, to be just jumped over; and (3) permute the order of local variables on the stack, static variables, and the routines of dynamically linked libraries and of the program itself. Such obfuscations make exploits that rely either on absolute addresses (stack smashing, return-into-libc, heap overflows, double free, data modification) or on relative addresses (partial overwrites and data modification) much more difficult. Many attacks are very unlikely to be successful when this method of obfuscation is used (the authors report a $4*10^{-5}$ success probability with return-into-libc attacks). It also involves no change to source code, and very little overhead.

High Coverage Detection of Input-Related Security Faults

Eric Larson and Todd Austin, University of Michigan

Many software vulnerabilities, such as the latest MS RPC DCOM vulnerability, arise from the failure to perform proper boundary checking on data. Data received from a network, for example, is often trusted and put into a buffer without first checking that the buffer is big enough. Other bugs occur when string library functions are improperly used, and these bugs can lead to serious security vulnerabilities, such as stack overflows and format string bugs. Searching for these bugs can occur at several different stages. It can be done at compile time, in which case there is no dependence on what input is actually used, but this makes things like keeping track of data on the heap difficult. Instead, the authors decided to take the approach of checking at runtime, but eliminated many common weaknesses of doing it at this stage, such as specifying actual input to check for bugs. Since the method will find a bug even if a dangerous value is not input, it only needs to be used during the software testing phase. When testing is complete, it is no longer used, and therefore there is no performance penalty. The authors' method involves shadowing any variables that contain user input. For example, integers are shadowed by a variable that stores its lower and upper bound, strings are shadowed by a variable that stores its size and whether it is null-terminated, and array references are shadowed by a variable containing possible ranges. These shadow variables are adjusted at any control points, such as loops, if statements, and arithmetic operations. At any use of the input that is potentially dangerous, the possible range of the input, not the input itself, is checked, so even if a dangerous value is not input errors can be detected. Using their implementation, the authors tested eight different programs and discovered 16 bugs, including three in the popular OpenSSH program. Most of these were the result of unbounded integers used in loops. Their method does have some limitations, such as only checking the execution paths taken during runtime, and it also has high runtime performance penalties. Some of their future work will involve optimizing their current implementation.

INVITED TALK: WHEN POLICIES COLLIDE: WILL THE COPYRIGHT WARS ROLL BACK THE COMPUTER REVOLUTION?

Mike Godwin, Public Knowledge

Summarized by Tara Whalen

Mike Godwin described how legislation put forward by content providers (media interests) will impede progress in the computer sector. The fundamental issue is that computers are very good at making copies: this is part of their basic functionality. This is not news to the IT sector, but it was news to the media (particularly the music industry). Content companies did not expect duplication of digital media; the music industry is an object lesson for the rest of the content industry (TV, movies) – they are looking at file trading, fear it will destroy them, and want to stop it.

In the summer of 2000, Michael Eisner told Congress there was a need for more legislation to protect copyrighted works. Only two years earlier, Jack Valenti said, "We aren't going to need legislation, because DCMA will protect us." But Eisner felt that more legislation was necessary, stating, "The problem for us is computers." Not file trading, or broadcasting, but computers. As usual, media players wanted targeted legislation, which is never as narrow as they think it is.

Hollywood can't say, "We have to stop the computer revolution." This won't work at all. But they can propose systems that have the effect of slowing down the pace of the computer revolution. For example, in the mid-1990s, Hollywood proposed legislation to protect video content through a scheme that marked every small set of frames. You'd play the video and the computer would look for the marks – if marks not there, then okay to copy, else abide by copyright rules. This is a very inefficient solution from a computer perspective: You need to dedicate resources to check for copyright marks. After extensive IT and media debate, they eventually derived the DVD standard. This standard (a) doesn't need to look for a mark, and besides which (b) DVD turned out to be a huge windfall.

Godwin went on to say that this experience taught different lessons to IT and media. IT companies concluded that if they negotiate with Hollywood, they can get a good compromise. What Hollywood concluded is that if they muscle hard enough, they can drag IT to the table and get largely what they want. Fast-forward to the Hollings Senate bill. It would have required a copyrighted-work-detecting chip built into every digital device. For this approach to work, there would need to be a regulatory buttress supporting it, because you can beat this scheme fairly easily. This now moves the debate into the arena of policy decisions, which would require re-architecting the digital world from top to bottom.

Godwin is concerned that such legislation will hamper progress: IT has been built on open architectures, which created opportunities and investment. "If Hollywood had its way, computers would be more like consumer electronic devices, with limited, controlled functionality. . . .What's troubling to me is that the content guys don't seem to see what harm they may be doing. They know that marking schemes require making devices untamperable, but they don't seem to realize this has a high cost – to us, but also to them. They will slow down the computer revolution that they themselves use for their business."

Godwin added that this is about making computer platforms acceptable to the content industries. We value user control in the IT community, and we don't want to give that up, but this model is now at risk. He concluded his talk by asking: What do we tell Hollywood? The glib answer is, "Develop another business model." A better answer is, "If you are concerned about content, we can design a more secure system that is leaky but overall will still make you money [like the DVD model]."

The talk was followed by extensive audience commentary and questions:

Q: We seem to be losing the Hollywood battle. What position can we actually hold?

A: The best compromise is probably to tell Hollywood to encrypt content at source and restrict the decoding to software decoding, such as with the DVD model.

Q: But studios aren't happy with the DVD model – can we hold them to it so they won't keep demanding changes?

A: When we negotiate compromises, we need to include consumer interests, not just studio interests.

Q: This situation reminds me of the crypto wars, with lots of regulation set up against a small group of geeks. Progress went ahead only when business came forward as an ally – who's the new ally in this arena?

A: I'm happy to work with any company who wants to work with me on these proposals. There are IT people who are suspicious of Microsoft or Intel, and you shouldn't assume that they are your enemies: use their muscle.

REFEREED PAPERS: DETECTION

Summarized by Chris Ries

STORAGE-BASED INTRUSION DETECTION: WATCHING STORAGE ACTIVITY FOR SUSPICIOUS BEHAVIOR

Adam Pennington, John Strunk, John Griffin, Craig Soules, Garth Goodson, and Gregory Ganger, Carnegie Mellon University

Current intrusion detection systems are often implemented at either the host level or the network level. There are scenarios, however, where both of these can fail. If an attack is too new and there is no signature for it, or worse yet, if it is misconfigured, a network intrusion detection can fail to spot an intruder. Host intrusion detection systems can also be disabled by rootkits after a host has been compromised, and logs can be scrubbed to cover the attacker's footsteps. Adding storage-based intrusion detection could help "augment the capabilities" of already existing systems and spot some of these attacks. The authors' idea was to implement an intrusion detection system on a host used for storage by multiple computers. With this setup, the intrusion detection system sees all persistent activity, and is also on a separate self-contained host, so it is not susceptible to being disabled after one of the hosts that uses it is compromised. It monitors changes to static files or corruption of well-understood files such as /etc/passwd, unexpected changes to the middle of a log file, or any suspicious content. The authors state that the storage IDS is no "silver bullet" and has limitations and weaknesses like any other IDS. One unique limitation is that it only sees storage traffic, so other traffic such as denial-of-service attempts will not be detected. It is also susceptible to general IDS weaknesses, such as false positives and misconfiguration. It does, however, add another level of detection. The authors concluded that storage IDSes provide a new vantage point to watch from, with minimum space and performance costs.

DETECTING MALICIOUS JAVA CODE USING VIRTUAL MACHINE AUDITING

Sunil Soman, Chandra Krintz, and Giovanni Vigna, University of California, Santa Barbara

One of the original goals of Java was to provide dynamic content embedded into a Web page. Since then its use has spread much further, due to such strengths as security, flexibility, and portability. Currently, Java security consists of type system and verification mechanisms, as well as mechanisms that can be used for authentication and access control. Fine-grained auditing and intrusion detection could improve the existing security built into Java, but running a host-based IDS as a separate process will not be effective, since multiple Java applications are run within the same virtual machine. The goal of the authors was to add an event stream at the JVM level with fine-grained response, which would allow for intrusion detection within the JVM. To create their built-in event stream, the authors first began with JikesRVM and extended it to associate a user ID and IP address with each thread. They also built in an auditing system that consists of an event driver, an event queue, and an event logger thread that runs as a system thread and reports to an external audit log. This audit log is then processed by an external IDS based on the STAT framework developed by the authors in a prior work. They extended that STAT core with a language-extension module, an event provider that collects the events, and a response module for reacting to attacks. To test their IDS they created various attack scenarios and showed the reports generated from them. They demonstrated harmful thread intercommunication, unauthorized access detection, and privileged information leakage. The alerts generated from these attacks included time, action, source thread ID and UID, internal network and remote address, sensor that detected the attack, result of the attack, and message-specific data. They also tested the performance with both partial and full logging enabled: performance time increased from 1 to 2.5 seconds (on an Intel Xeon 2.4GHz with 1GB RAM). In the future they plan to reduce this overhead.

Static Analysis of Executables to Detect Malicious Patterns

Mihai Christodorescu and Somesh Jha, University of Wisconsin

Mihai Christodorescu began by discussing how malicious code detection, like many other areas of computer security, has really become an arms race between the good guys and the bad guys. Detection of viruses and other malicious code began with the use of signatures to identify the malicious code. To foil this detection, the bad guys started using techniques such as register renaming, and the good guys countered this by using regex signatures that were essentially templates that allowed the signatures to have gaps. Next, the bad guys started packing and encrypting the malicious code, and the good guys countered this by using emulation and heuristics. Now the bad guys have started to use code reordering and integration to try to keep their malicious code from being detected, and this type of obfuscation is not always detected. The authors obfuscated four viruses using NOP-insertion and code transposition, and then scanned them using three popular antivirus tools. The viruses that they used were Chernobyl, zombie-6.b, f0sf0r0, and Hare, and none of the scanners detected the obfuscated versions. Their tool, however, called SAFE (Static Analyzer for Executables), detected all four of them. A malicious binary is first loaded into SAFE, then a control flow graph (CFG) is created for each procedure. A program annotator then reads a CFG and a set of abstraction patterns from a library. The output is an annotated CFG that has patterns associated with each node of the graph. For example, a NOP instruction would have a pattern matched with it that states that it is an irrelevant instruction. A pattern consists of a list of typed variables, a sequence of instructions, and Boolean expressions. A detector then reads this annotated CFG and compares it to a malicious-code automaton to decide whether it contains any malicious code. The advantage of their approach is that SAFE is able to detect malicious code even if it has been obfuscated. As was pointed out during the Q & A, as long as there is a malicious-code auto-maton for that type of behavior, obfuscation will not fool SAFE. The performance of their implementation seemed to be successful, as no false positives or negatives were encountered. They plan to improve SAFE by having it look for different types of malicious code, such as trojans.

INVITED TALK: PHYSICAL SECURITY: THE GOOD, THE BAD, AND THE UGLY

Mark Seiden, MSB Associates

Summarized by David Molnar

Mark Seiden began by talking about the basics of physical security. In the physical world, unlike online, the concept of "secure perimeter" is real and meaningful. Security audits begin by talking about security analyses. Unfortunately, it's important to make sure that a security analysis is realistic. Too often, whenever someone says, "I did a security analysis and determined that it was not a threat," "it" ends up being something they didn't think about.

Seiden went over the basics of establishing a secure perimeter. Check doors, windows, and also roof and tunnel access. As an undergraduate at Columbia University in the late 1960s, he reported on student demonstrations and police reactions to them by sneaking into the main library at Columbia through the sewers. Because the police had chained and locked the front door, they thought it was "secure."

Another case dealt with a supposedly "ultra secure" co-location facility Seiden had visited as part of his work. The facility boasted motion detectors, alarms, and other measures for notifying security when a breach had occurred. Unfortunately, it also had pull-up floors. Seiden and a colleague attempted to trip the response system by first going under the floor to come up in a server room and then tripping the motion detector. Instead of calling security, the motion detector simply opened the locked door, a common setup to prevent someone from accidentally being locked inside.

Seiden also talked about the divergent security philosophies of the physical and network security communities. For the physical security community, security by obscurity is valid – it means an adversary has to put in more work to determine the layout of your installation, and this may buy you time after a break-in. By contrast, the network security community does not value security by obscurity nearly as much and therefore will share information at conferences such as this one. As a concrete example, the speaker pointed to Matt Blaze's recent work on rights amplification in master-keyed locks. The physical security community had known about this issue for a long time, but their culture was to keep it secret and not talk about it. The network security culture published it so as to effect systematic change.

Another side effect of the culture of secrecy is the suppression of information regarding incompatibilities that affect security. For example, the most popular deadbolt and the most popular electric strike (a device that allows electric operation of a lock, such as by a computerized access control system) are incompatible. Installing the strike disables the deadbolt-locking mechanism, degrading the lock to the security of just an ordinary house lock.

As a case study of the boundary between computer and physical security, the speaker talked about a magstripe access control system he had audited. The system consisted of panels and a central access control list system. Panels had limited memory and communicated with the main system via dialup modem. The system then pushed lists of approved users and pulled access events (such as attempted unauthorized access). Unfortunately, the dialup connection was not authenticated in any way; anyone could call up the panel or alternatively interpose and pretend to be the main access control system. At the least, such a compromise could be used to flush the panel's record of access control events, because panels did not keep records after the first attempt to send events to the main system. At the worst, password guessing or eavesdropping could be used to take complete control of an access panel.

The speaker and his team also took a look at the source code of the access control system. It turned out that the source code contained #ifdefs specific to each customer of the system. From the code snippets, much could be inferred about the structure of these customers' access control needs. The audit also uncovered several cache-consistency issues between panels and the main system. When questioned, the manufacturer claimed that everything was operating "exactly as designed."

REFEREED PAPERS: APPLIED CRYPTO

Summarized by Gelareh Taban

SSL Splitting: Securely Serving Data from Untrusted Caches

Chris Lesniewski-Laas and M. Frans Kaashoek, MIT

This presentation focused on the problem of reducing bandwidth load from a server Web site while still allowing high-content throughput to its clients. One possible solution is to use mirror sites. That is, the server can ask a group of volunteers to proxy the contents of the site and so distribute the bandwidth load. However, this scheme has potential for mischief, whereby an adversary can pose as a volunteer and serve modified contents to the clients. So this scheme requires trust in the proxy from both the client and the server.

A cryptographic solution is for the server to attach a signature to the data being served and allow the client to ensure the integrity and authenticity of this content by verifying the signature. Existing protocols that deal with this problem are not practical, since they either require the client to use a specialized browser (e.g., S-HTTP, SFSRO), thus leading to the depreciation of their deployment, or the protocol operates at the channel level and not the file level.

An example of the latter protocol is the widely deployed browser support for the SSL protocol. The author splits the server end of the SSL connection by introducing a proxy between the server and the client, such that the proxy is not privy to the shared client-server key. In this scheme, the proxy caches the data content of the server. In response to a request from the client, the server sends the proxy the unique identifier of the requested data as well as a message authentication code (MAC) for the data, using the shared secret key. The proxy then sends the cached data and associated MAC to the client. The server thus is able to offload bandwidth to mirrored sites while maintaining the integrity and authenticity of the data. The problem with this scheme, however, is that there is no end-to-end confidentiality; the server only distributes bandwidth load and not CPU.

In short, SSL splitting reduces bandwidth load while guaranteeing end-to-end data integrity and offering transparency to the client. For more information, visit *http://pdos.lcs.mit.edu/barnraising/*.

A New Two-Server Approach for Authentication with Short Secrets

John Brainard, Ari Juels, Burt Kaliski, and Michael Szydlo, RSA Laboratories

Ari Juels opened his presentation by recalling the other name of the project, "Nightingale," explaining, "It was a midnight project." He went on, "The project began as an inquiry into the mind of the hacker." What does a hacker want? Fame, wealth, love . . ? Alas, the hacker wants "root access"!

The sensitive data the project "Nightingale" is intended for are short secrets and "life" questions, commonly used on the Internet today to authenticate users. How are these data stored securely on the application server? There are two main types of protections: frontline defenses such as up-to-date security configuration, authentication, and intrusion detection, and cryptographic rearguards such as hashing and encryption. However, due to the architecture of the scheme, there exists a "single point of compromise." This means that once access is obtained to the server, all data on the server is compromised. The adversary is able to launch various attacks, exploiting the weaknesses of the cryptographic techniques employed.

Nightingale eliminates the weaknesses of a single point of compromise by distributing the secret data between two servers, the application server and the Nightingale server, so that compromise of a single server does not compromise the data. The data can be shared using threshold cryptography and can be used to perform distributed (or blind) secret verification, remote reconstruction of data shares, or management of cryptographic keys.

The architecture of the system is important. Instead of allowing both servers to be equally accessible by the client, it is proposed that the Nightingale server be placed behind the application server, allowing better security protection as well as client transparency of the servers. Furthermore, different implementations of the servers allows different levels of security and protection for the secret data.

Presently, Nightingale is being integrated into the RSA Security Inc. product lines. For more information on the Nightingale system, please refer to *http://developer.rsasecurity.com/labs/nightingale*.

Domain-Based Administration of Identity-Based Cryptosystems for Secure Email and IPSec

D.K. Smetters and Glen Durfee, PARC

How do we make public keys more usable?

Deployment of cryptographic techniques today in such applications as secure email and IPSec has been hampered by the distribution of keys in a public key infrastructure (PKI). More specifically, the sender has to ensure that the receiver has generated a certified key pair, and the sender must also obtain an authenticated copy of the receiver's public key. This means that trust must exist between the sender and the certificate authority (CA) of the receiver. However, there are many difficulties involved in establishing a large-scale PKI, most importantly the question of large-scale trust between users.

The authors automate key distribution in a PKI setting by combining the idea of limited trust in the existing hierarchical DNS server infrastructure (using DNSSec to ensure security) with the usability advantages of identity-based cryptography (IBC). The main idea of IBC is to make the private key a function of the public key and some IBC parameters, thereby allowing the "identity" of a user to be her public key. In practice, however, the global scope of trust and namespace needed in a large-scale infrastructure is unacceptable for most applications. The proposed DNS-based IBC (DNSIBC) allows limited but usable scope.

The authors bootstrap trust from DNSSec such that clients are authenticated in their own local domain. This means that key-escrow, an automatic side effect of IBC, will also be limited to the local domain. The scheme requires no secure servers on the Internet, and only IBC parameters need be stored on DNS. Key revocation can be easily implemented by using time expiry for the keys. DNSIBC is easily deployed and incorporated in S/MIME and IPSec.

INVITED TALK: THE INTERNET AS THE ULTIMATE SURVEILLANCE NETWORK

Richard M. Smith

Summarized by David Molnar

From the abstract, I expected that the talk would be about the current uses of the Internet for surveillance. Instead, the speaker gave a more speculative talk about how the Internet could be used in the future to enable global tracking and surveillance of individuals. This speculative talk managed to provoke a great deal of discussion, particularly in the area of radio frequency identification (RFID) technology.

Smith began by pointing out that more and more devices will become IP-enabled, including phones, laptops, and PDA devices. He then defined devices analogous to a URL but intended to encapsulate information about a target's physical location. These location markers could be generated and then sent surreptitiously through any Internet-enabled device to a central database. The result would be a pervasive infrastructure for keeping track of the locations of people and passing the information back through other devices to a central database.

How can a device recognize what person has come in contact with it? The speaker pointed to several methods, but the method that excited the most comment was that of keeping track of a person by his or her personal RFID profile. The speaker outlined the current state of RFID technology and future trends.

Passive RFID tags consist of small chips connected to antennae. The chips are powered by a radio broadcast from a special RFID reader. These passive RFIDs carry an ID number and not much more, which allows them to be used to replace barcodes and to perform remote tracking. Currently, passive tags cost less than 50 cents (US) per tag, but the eventual goal is to reduce costs by another order of magnitude, to five cents, to enable per-item RFID tagging. Active RFID tags are more expensive, have their own internal battery, and are generally used for containers or other aggregates of goods.

One of the envisaged applications for per-item RFID tagging is theft reduction. The speaker talked about a recent field trial of RFIDs in Gillette Mach3 razors. Mach3s are easily resellable and, consequently, often shoplifted. The trial combined RFIDs on each razor package with a "smart shelf" that took a picture of a customer whenever he or she removed "too many" packages of razors. The idea here is that if the razors are later found to have been shoplifted, there is a picture of the perpetrator.

Smith then demonstrated a Texas Instruments RFID reader and invited audience members to come up afterwards to check whether their devices carried RFID. The reader connected to a laptop; associated display software picked up RFIDs in several items carried by the speaker and displayed them on the screen. The audience could then clearly see how some innocuous-looking items from the speaker's wallet in fact turned out to contain RFID tags.

The discussion of RFIDs turned out to be controversial even before the question and answer session. Perry Metzger interrupted the speaker at several points to underline potential threats involved in pervasive RFID. For example, would it be possible to build a device that allows someone to travel through a crowd and read off the RFID information of all passersby? Metzger claimed such a device was easy with current technology, while the speaker was not so sure. The speaker also disagreed with Metzger about the feasibility of an active device that could "burn" out RFIDs by sending them too much power.

During the question and answer session, much discussion focused on the nature of RFID technology and its implications. Some audience members questioned the speaker's assertion that per-item RFID tagging would be driven by shoplifting protection, since shoplifting is just "part of the cost of doing business." Others wanted to know if the RFID reader demonstrated would catch all RFIDs that might be on their person; the speaker replied that was not true, because multiple standards for RFID currently exist. A reader for one standard may not read tags for another. A great deal of discussion involved the range of RFID readers and the possibility of RFID "burners," but without much agreement.

Summarizer's note: Readers who desire more information on RFIDs may find the following links helpful:
Silicon Valley RFID Yahoo! Group: *http://groups.yahoo.com/group/sv_rfid/*;
MIT Auto-ID Center: *http://www.autoidcenter.org/*
RFID Privacy Symposium at MIT, November 15: *http://www.rfidprivacy.org/* and associated Web log *http://www.rfidprivacy.org/blog*.

PANEL: REVISITING TRUSTED COMPUTING

Panelists: Douglas Barnes, Dave Safford, William Arbaugh, and Peter Biddle

Summarized by Catherine Dodge

The panel began with short comments from each of the panelists: Peter Biddle from Microsoft, Dave Safford from IBM, William Arbaugh from the University of Maryland, and Douglas Barnes founder of the Government Open Technology Information Project.

In his comments, Biddle referred to a technology as NGSCB (pronounced ink-scab), its current moniker within Microsoft. In his presentation, he focused on how NGSCB is being developed to meet customer concerns about the erosion of their IT security perimeter. Increasingly, mobile computers, cell phones, PDAs, and other wireless devices are accessing the network, often bypassing outdated security measures not designed to handle these new technologies. He cited the occurrence of Xbox tournaments over the Microsoft corporate network as evidence that administrators have less and less control over the ways their IT systems are being used. This has led to customers wanting a means of doing application-to-application authentication that will protect intellectual property information, be it a clinical trial database or a secret recipe. Along with this protection, customers are asking for technologies that will allow them to share select information and systems with suppliers, partners, and customers in a secure and controlled way. Micro-soft's vision is that applications must have third-party verification before running, which would prevent an application that has had a keystroke logger added to it, or a bogus user account, from running. Biddle's final point was that Microsoft has heard loud and clear that customers do not want to be "locked in," having once deployed this technology. For further reading, consult a paper on authenticated operation of operating systems (*http://cs-people.bu.edu/mpe/acisp.pdf*) and white papers about the NGSCB architecture (*http://www.microsoft.com/resources/ngscb/productinfo.mspx*).

Dave Safford began his comments by holding up the August 2003 issue of *Linux Journal*, which includes an article on the open source tools for TCPA that IBM has developed. His perspective is that no matter what the vendors say, nothing can convince a skeptical user base that TCPA isn't "evil," therefore any solution must be open source. He noted that the current working document (*http://www.trustedcomputergroup.org*) for the TCG consists of over 320 pages of dense language. Because of this, most people engaged in the debate over TCPA have not actually read the specification. The Trusted Platform Module (TPM, or TCPA chip) upon which the platform is based is essentially an RSA chip, with the key never leaving the chip. A diagram of the chip can be found in the *Linux Journal* article. Safford kept emphasizing that solutions to the trusted computing problem should and will be open source. Further information, including white papers and source code for a TPM device driver, can be found at *http://www.research.ibm.com/gsal/tcpa*.

Prof. William Arbaugh drove home the point that any new technology can have unintended or dual uses, a perspective which placed him fairly middle-of-the-road in the debate. While everyone lumps all trusted computing platform concepts together, in his view this is incorrect. There had been very emotional debate around these platform proposals, namely, concerns voiced by users that they will become "locked-in" to the platform, that free software will be excluded from the playing field, and that the privacy of users will be seriously violated. While a trusted platform could support such usage, the technology alone cannot do these things. The issues raised above result from the policy that the trusted platform would enforce, not from the platform itself. And just as there are potential drawbacks, there are also many potential benefits, such as protected storage and proof of configuration to a third party. Ultimately, users should have the choice of what kind of policy to use – and a policy language and interface that enables them to understand the policy choices they make.

Douglas Barnes gave his perception of what life would be like once a TCPA-like platform became prevalent. In his view, the technology will amplify existing market power, while ending the sense of "ownership" users have come to feel over content and software. He also views any espoused "choices" as an illusion, since third-party validation cannot be done with just anyone if the second party (namely, the one constructing the TCPA) does not allow the user to communicate with them. To ensure that this is not what the future of trusted computing looks like, Barnes urged users to demand that their own benefits take precedence over the more profit-driven benefits to be reaped by commercial parties. Users should also not feel pressured to buy until proper safeguards and assurances are in place. The discussion that followed was mainly fueled by questions around who will ultimately control how a trusted platform is utilized – the vendors or the user? Repeatedly questioned as to how we can be sure that Microsoft won't "do it," interpreted to mean that the company will not suddenly limit the types of policies that can be implemented on NGSCB, Biddle stated that Microsoft would be bound by the contracts it had entered into regarding the NGSCB. They would likely be bound by the most strict of the contracts they had signed. Pressed on development aspects of the NGSCB project, Biddle provided some details. Microsoft is writing the code in C and has people doing formal methods proofs for the project. They will not prove the entire code but will evaluate key components, such as the memory management module. Additionally, the team maintains two development trees, one for code yet to be examined and validated and one for production. The development team is also utilizing tools that do not allow changes to the code interface without making the appropriate changes to the specification. Biddle emphasized that if they do not end up with a system that is comprehensible by a single individual, they will consider the project a failure.

Another interesting point brought to light was how humans interact with security. Most panelists agreed that it is a problem without a good solution at this point. Some technical details of the TCPA were also covered. The panelists noted that the system is not designed to be secure against a physical attack. Safford said that IBM's goal is to get the TCPA platform to interact with virtual machines. Many in the audience were also concerned about how third-party software would be able to run on the platform. Biddle said that the default setting would be to allow any software to run. This can then be configured by the user to narrow down the kinds of credentials the user accepts. His comments indicated that the issue of how a small software developer can "certify" their software comes down to how the majority of users configure their security policies under the TCPA.

REFEREED PAPERS: HARDENING II

Summarized by Clif Flynt

PREVENTING PRIVILEGE ESCALATION

Niels Provos, Peter Honeyman, University of Michigan; Markus Friedl, GeNUA mbH

Two problems with secure computing are bad design and bad implementation. Even with a good, secure design, a program that has a buffer overrun or other implementation error can be compromised. If that program runs in a privileged state, it opens the door for all kinds of trouble. Provos described an application architecture that reduces the potential for trouble by separating those sections of an application that require privileges from the rest of the application. This reduces both the amount of code that must be carefully examined to create a secure application and the impact of implementation errors in the application. For example, applications that perform user validation require access to files that users are not allowed to read (/etc/shadow) and thus must be run as SUID root.

Provos described how the OpenSSH application can be split into a small monitor section that runs with privileges and a larger slave section that performs most of the interaction with a user. The downside to this is that there is no automated way to split an application into privileged and unprivileged sections. Provos' implementation of this concept separates the monitor and slave into two applications, which use a socket to exchange requests and data. He explained that processes under UNIX are protected entities, and only the owner can send signals, debug, or otherwise interact with a process. Requests a slave may send to the monitor include Information (request challenge, provide response from remote, monitor compares) and Capabilities (may access file system for slave). When a slave dies and a monitor creates a new slave with a new ID, there may be state information that must be saved and loaded into the new slave. In this case, the slave exports state to the parent before terminating, and the new slave imports state without the monitor evaluating the data. In practice, exporting the state is pretty messy. This uses XDR-like data marshaling for Global Structures; to handle dynamically allocated state, the application uses shared memory. The master keeps the shared memory open and lets a new slave attach to shared memory. Provos noted that this required a new malloc to allocate memory in shared mem space instead of normal heap. The reimplementation of OpenSSH using this technique included a list of permitted requests; if a slave's request is unrecognized, the master immediately terminates the slave. In this implementation, the slave owns the SSH session socket and sends Information requests to the master for server signature, testing password validity, a challenge to send the SSH client, and to check the SSH client response. Thus, all determination of identity of client is done in the monitor; if a malicious user can manage to compromise the slave with a buffer overrun or a similar attack, the application won't allow malicious login.

IMPROVING HOST SECURITY WITH SYSTEM CALL POLICIES

Niels Provos, University of Michigan

Niels Provos discussed techniques for preventing applications from performing unexpected actions by reducing their access to system library functions. He pointed out that it's not possible for anyone to have intimate knowledge of all the applications running on their computer. Even given source code, and assuming enough time to examine it completely, there's no guarantee that the running application is actually compiled from that code. We must assume that any vulnerability in a system is known to an attacker, even if it's not known to the user. Any damage to a system must be done through the system call library. Intercepting these calls will prevent a malicious program from modifying a file system, overwriting memory, invoking other programs, or other unpleasant actions. Provos's work resulted in the Systrace system, which catches all system function calls on the fly and determines which calls are allowed (or require permission) to an application. When using Systrace, no applications need to be run with root privileges (SUID root). Instead, applications that require privilege escalation to invoke system functions at a superuser level are granted access to the only necessary subset of the system functions, without exposing other functions to possible abuse. Provos solved the problem of rule complexity by putting a user-friendly front end on Systrace that pops up a query when an application tries to do something the rules don't allow. If the user is willing to allow this operation, a new rule is generated and added to the rule set. With this technique, one can start with a fully locked down system that can't be used for anything, and develop a set of rules that permit work to be done quickly. The Systrace system is implemented as a hybrid of kernel and user-space functions to provide for portability and efficiency. Provos notes that a good policy is one that allows only the actions necessary for the intended functionality of the application and denies everything else. We can generate policies automatically, or a user can define policies interactively via a GUI dialog. In practice, he found that policies with a subset defined, and dialogs when something unknown comes in, converge to good policy fairly quickly.

INVITED TALK: THE INTERNET IS TOO SECURE ALREADY

Eric Rescorla, RTFM, Inc.

Summarized by David Molnar

Note: Slides from the talk are available at *http://www.rtfm.com/TooSecure-usenix.pdf.*

Despite nearly two decades of open research in security and cryptography, the state of real-world system security does not seem to have improved much. Why is this and what can we do about it? Eric Rescorla placed the responsibility for this state of affairs on an incorrect threat model and lack of attention to real-world security problems. Practical software security is much less glamorous than coming up with a new cryptographic hole. As a result, perhaps we are focusing on the wrong areas of research for practical protocols.

The rest of the talk developed this point with specific examples. The speaker considered the SSH, SSL/TLS, IPSec, PKIX, S/MIME, and WEP protocols. These were divided into "wins," "draws," and "losses." SSL/TLS and SSH count as "wins," IPSec, PKIX, and S/Mime as "draws," and WEP as a security "loss."

In each of these protocols, the threat models assume that endpoints are inviolate and links are completely controlled by an adversary. The speaker then argued that in the real world, most of today's problems come from endpoints compromised by virus, worm, or DDoS attack. Therefore these threat models are not well matched to today's real vulnerabilities in protocols. In particular, the practice of requiring public-key certificates and certificate authorities to protect against man-in-the-middle attacks adds significant complexity to deployment.

In SSL, this complexity is mitigated because only servers need certificates, but in the case of encrypted email, the requirement seems to have significantly hindered adoption. The speaker also made the point that perhaps people don't actually want encrypted email, despite the fact that everyone says they want encrypted email and there are plenty of startups past and present for the concept. After all, we are on our third generation (counting from MOSS) or more, and still no one uses it.

The SSH "leap of faith" model, in contrast, was held up as a good example of a usable model that manages to resist most attacks. The speaker noted that SSH key management was considered a very bad idea at the time, but now everyone thinks it's the right way to go!

Rescorla then discussed the difference in credit and glamour for flaws in cryptography and flaws in software quality. He made the point that we will review the entire protocol to deal with one padding attack that has never been implemented, as far as anyone knows, but that we seem to invest comparatively little work in dealing with buffer overflows or usability enhancements. For example, in 1998, Daniel Bleichenbacher found a vulnerability in the padding used by SSL; his name is well known, despite the fact that his attack has never been seen outside the lab. In contrast, the discoverer of the buffer overflow exploited by the Slapper worm, which compromised thousands of servers, is practically unknown. This difference in credit leads to a difference in research emphasis. More recent examples of "theoretical" holes in cryptography that were given great attention included Serge Vaudenay's padding attack on CBC and Phil Rogaway's attack on padding.

Many of these theoretical attacks can be prevented with good "crypto hygiene"; for example, timing attacks are resisted by using the blinding code in OpenSSL. Operational solutions such as this can avoid the need to upgrade the entire protocol to deal with a weakness. This is important because the protocol revision process is time-consuming and often bogs down. Necessary functionality improvements are held up because security issues are considered in the same committee. A secondary point was made about the efficiency (or lack thereof) of standards committees; for example, SSH was introduced to the IETF several years ago and is only now becoming standardized, IKE is two years overdue for a rewrite, etc.

The speaker ended by discussing end-user requirements for security. For example, when they say "security is important," they tend to mean "I need to know what to tell my boss." This led to a discussion of methods for fixing our security research practice and getting market feedback on what is important to fix and what is not. Finally, he left the audience with a list of questions for shaping future research in security.

One audience member commented that he was from the NSA, and that in the classified community they had paid a lot of attention to issues of software quality influencing the reliability of the transaction. The key question is *what* you are relying on the software to do. The audience member commented that attestations of software quality will come to play a greater role in protocols; parties may require certificates stating that the software of the other party has been audited to an acceptable level before entrusting data.

Another audience member commented that the speaker's view of cryptographic considerations in the design of security protocols might be colored by the specific issues of the Internet Engineering Task Force (IETF), where many of the examples originated. The speaker replied that he had seen similar issues outside IETF and in other organizations.

David Larochelle mentioned that SSH also added functionality with regard to xterm forwarding, which helped it dominate its market segment. He then referenced his paper on the subject (*http://www.cpppe.umd.edu/rhsmith3/papers/Final_session3_farahmand.navathe.sharp.enslow.pdf*).

David Molnar asked if the speaker had a good place to learn about crypto hygiene. The speaker replied he was too busy to write a paper on the subject but invited others to do so. "Maybe you can do it."

REFEREED PAPERS: THE ROAD LESS TRAVELED

Summarized by Scott A. Crosby

SCRASH: A SYSTEM FOR GENERATING SECURE CRASH INFORMATION

Peter Broadwell, Matt Harren, and Naveen Sastry, University of California, Berkeley

This presentation dealt with crash dumps. When a program crashes, it may leave behind a dump of its stack, variables, and memory contents to aid in debugging. Many popular programs are now including automated reporting so that users can easily send these dumps to developers. Such dumps are valuable debugging and development aids, but may contain sensitive information such as passwords, bank account numbers, keys, trade secrets, or PINs. Users may want to send these dumps to developers but don't because they don't want to disclose their secrets.

This paper outlines how to make still useful crash dumps available to developers without disclosing user secrets. The approach used is to annotate some of a program's variables as "sensitive" and then to apply a data-flow analysis from CQual to identify all other variables that may become tainted with sensitive data. Those variables are also marked as sensitive. A custom malloc library that supports region-aware allocation is used to store sensitive data in one region, separated from nonsensitive data, which is stored normally. When a crash dump happens, the sensitive region is wiped so that it won't contain any data that was tainted as being sensitive.

In the performance analysis, a microbenchmark implementing a recursive GCD solver shows a 22% degradation, while the time taken by scp to transfer 100 megabytes of data degraded by 6%. They also tested two interactive programs, the GNOME calendar application and a Palm Pilot synchronization program. For these applications, under 25% of the stack variables were marked as possibly containing sensitive information and no impact on performance was observed.

Implementing and Testing a Virus Throttle

Jamie Twycross and Matthew W. Willia, HP Labs, Bristol

Jamie Twycross started this talk by showing the infamous movie of the Sapphire worm infecting tens of thousands of hosts in minutes. He then reminded us that patching systems don't work; patches can be available for months and yet tens of thousands of computers may remain unpatched and infectable. Signature-based worm detection also doesn't work. Worm signatures – e.g., ports they scan – require prior knowledge or active infection to determine. By that time it is too late.

This paper proposes a different scheme: detecting a worm based on its network behavior, in this case, outgoing traffic. The response to detecting adverse behavior is benign – it delays traffic but does not drop it. The detector is based on the idea that most hosts will only regularly contact a small working set of hosts; worms will contact many hosts. If a host attempts to contact a host that is in the working set, the connection goes through immediately. The working set size is set to five hosts. Otherwise, the connection is delayed. The host can assume it is infected if the queue of delayed connections grows too long.

Because connections are delayed, this mechanism does not create false positives, and because it detects scanning behavior, it can work without knowing much about a worm. The authors showed that this scheme takes seconds to stop both simulated and real worms on their test network, and that only a slim portion of their normal desktop traffic was delayed at all.

Establishing the Genuinity of Remote Computer Systems

Rick Kennell and Leah Jamieson, Purdue University

This won the Best Student Paper award. Rick Kennel started his talk with a pronunciation guide to genuinity, then described why it is desirable. In current systems, forgery is a classic problem. Users can masquerade as other users, computers may be substituted for other computers, and simulators can masquerade as physical machines.

He gave as an extreme example remote NFS clients. NFSv3 depends on client-side trust to be a secure distributed environment. He asked whether clients can be trusted, including remote ones, on an insecure network with ordinary hardware and with no trusted human to look over the remote machine. He then described how an authority could be able to trust remote entities.

The authority can know if the entity is executing the correct code by requesting a checksum over it. Unfortunately, the remote host can lie. The authority can resist this by combining the checksum with a hardware test that can detect whether the checksum operation was actually run. Since computers can simulate other computers, a right answer is not enough. The authority must find a hardware test that is deterministic, but hard to simulate quickly and accurately, such as implicit parallelism, data-cache updates, or other aspects of the memory hierarchy. Once such a test is found, its response is regularly included as part of the checksum. The authority checks to make sure that the checksum is both correct and timely. Any attempt to virtualize hardware or alter the program is almost guaranteed to alter those hardware values and create a bad checksum.

Rick Kennell and Vern Paxson

INVITED TALK: THE CASE FOR ASSURANCE IN SECURITY PRODUCTS

Brian Snow, National Security Agency

Summarized by Catherine Dodge

Brian Snow spoke about the need for greater assurance in commercial off-the-shelf products. Snow discussed two distinct threat categories: generic and targeted. A generic threat, for example, is one posed by a burglar who wants to steal a VCR. The algorithm for such a task is relatively simple: find house; if house is dark, check door; if door is unlocked, go in; if locked, try next house. The subject of a generic attack must only invest in enough defense to make it cheaper and easier for the burglar to rob the neighbors. On the other hand, if a burglar were after your Picasso, the algorithm would become much more sophisticated: find house; if dark, check door; if door is unlocked, go in; if locked, break window and go in; if alarm visible on windows, rip off siding, punch through gypsum, rip out insulation, punch through dry wall and go in; if motion detectors present, cut phone line that dials the police upon alarm and go in via the above methods. The point is that defense against a well-funded, determined adversary is difficult and must be similarly well thought out and well funded. While the NSA would like to be able to recommend commercial products to its Department of Defense customers, Snow expressed his frustration that "I still haven't found what I'm looking for" and wondered "when will I be secure, nobody knows for sure" (via a video of song commissioned by Jim Bidzos for RSA '99, to the tune of the U2 song by the same name). Snow likened the state of today's computer security to cars built in the '30s, with no anti-lock brakes, no air bags, not even seat belts. Assurance, like an air bag, while generally invisible to the user, becomes prominent on failure. Snow extended the car analogy and recognized that there are not enough customers for GM to build tanks to sell at their dealerships. Yet he does wish that commercial products at least had the same level of security as a car built in 2003 does. While popular wisdom often says otherwise, Snow argued that quality and reliability can sell software. The example he gave was how Toyota and Honda moved into the US car market in the '70s, based solely on such attributes. Yet this kind of assurance cannot simply be slapped on top of a product at the end of the development cycle. Software engineering too often assumes a benign environment. Developers must move beyond this assumption and factor in malice at design time. Above all, Snow was concerned that in another 10 years we'll have more features but not more assurance, a state of affairs which led him to exclaim, "Am I depressed? Yes!" These features may include the ability to counter many hacker attacks, yet targeted attacks will still be overwhelmingly successful. "We will think we are secure, but we will not be."

After finishing this self-described "polemic," Snow outlined six areas in which vendors can address his concerns: operating systems, software modules, hardware features, systems engineering, third-party testing, and legal constraints.

Snow advocated improvements in operating systems, such as digital signature checks on modules prior to execution, utilization of the principle of least privilege, and enforcement of security policies. He also sees room for improvement in software modules. They need to be well documented, be created in certified development environments, and use separate design and review teams. Snow pointed out that a contractor with uncleared foreign national employees could apply good principles by having a German team design the project and a French team review it. Included in this process should be some degree of formal methods, as the tools to support their usage have gotten better. Hardware features are also essential to creating assurance, since they can create isolation in a way that software cannot. The major issue with hardware then becomes trust in the person you are purchasing from. Additionally, systems engineering is a key component of assurance. Questions that need to be answered in this arena include: How can products of unknown quality be used safely? How can components be composed in such a way that the assurance level is a synergistic result of the parts, and not be reduced to a "weakest link in the chain" situation? Third-party testing is another area that can help address assurance, since it provides something beyond "emphatic assertion" that a product provides the desired level of security. A current major failing of such certification processes is that the results are not easily understood by users. Ultimately, though, it may be legal constraints that convince software companies to invest in assurance. Establishing a fitness-for-use criteria and liability beyond the media that the software is distributed on would go a long way toward raising the bar in the software industry. And more insurers may do what some have already done, namely, charge different rates to insure a company based on which operating system and applications they use. Whether these changes really do come about depends on these three aspects: training, research, and assurance built in from the ground up.

Someone asked how those who develop free software can have their products certified, in light of the considerable expense this requires. Snow replied that there is no way of getting around the fact that a certification will cost money. He thought the best path for free software would be to create a foundation which would raise money specifically for funding the certification of such software projects.

WORK-IN-PROGRESS REPORTS

Summarized by David Molnar

Analysis of Electronic Voting System

Adam Stubblefield

Note: For more information, go to *http://www.avirubin.com/vote.*

The speaker discussed his and Avi Rubin's experiences auditing the source code of a widely used voting machine. Among the issues they discovered were that key management is handled by a single #define DES_KEY "XXXX" in the code, where "XXXX" is an ASCII string; this key appears to be the same across all instances of the machine. Another issue was that the code did not appear to have been audited at all for correctness or security; long stretches of code appear without any comments whatsoever. Audit features in the code were extremely weak; while changes do appear in an audit log, nothing seems to protect the log itself from being changed. In response to a write-up on these issues, the company claimed that the "voting system works exactly as designed."

One audience member asked if source code was still available. The speaker replied that it was the last time he checked.

Validator – Testing Firewalls

Clif Flynt

Flynt's framework for firewall testing, Validator, allows for defining "interactions" and "monitors" for firewalls. The framework also allows for the creation of "golems," autonomous agents that take scripted actions for firewall testing, and "probes," which test firewalls with known good or known bad packet traces.

The system is in late beta; it consists mostly of expect scripts. Interested parties are invited to contact the author at *clif@cflynt.com.*

Link Cut Attack

Steve Bellovin

Note: A draft version of the associated paper is available at *http://www.research.att.com/~smb/papers/reroute.ps*. As of this writing, the paper stated, "Please do not mirror or distribute."

The speaker opened by pointing out how we are close to deploying secure BGP and secure OSPF. These protocols address attacks where nodes lie to each other in advertisements about what routes they can provide. Unfortunately, there is an attack that these protocols will not address: cutting links to force routing protocols to route through adversary-controlled nodes.

Suppose that the adversary controls a node N that is in the same network as two communicating nodes S and T and that N is not initially on the path selected for routing between S and T. Further, suppose the adversary can cut a limited number of edges in the graph. Then the adversary can use some graph theory to figure out the edges to cut such that the resulting shortest path between S and T runs through the compromised node N. The graph theory is fairly simple, and the resulting algorithm runs quickly on graphs obtained from real network topologies. Because the attack changes the real topology of the network, mechanisms for preventing false advertisement have no effect. A worked example was shown in which an adversary caused the shortest path between two network nodes to run through the node it controlled.

Stream

Simson L. Garfinkel

URL: *http://stream.simson.net/*

The speaker raised the question, "Why, after all these years, do we not yet have encrypted email?" The answer lies in the cumbersome key management for PGP and the impossibility of obtaining S/MIME certificates. Therefore, the speaker created a new opportunistic encryption layer called Stream. The aim is for deployment as transparently and as widely as possible. Keys are placed in the headers, where they aren't noticed by mail agents. The speaker showed a plug in running with Eudora that talked to a Stream proxy responsible for key management and encryption.

One audience member asked if the software would be freely available. The speaker responded that it could be downloaded right now from his Web page. Precompiled binaries are available for FreeBSD, Linux, and Mac OS X; source is also available.

A Framework for Receipt Issuing, Contendable Remote Poll-Site Voting

Prashanth Bungale

The speaker presented a new design for an electronic voting system. A key feature of this voting system is receipt-freeness while maintaining voter verifiability. Voters can verify that their vote was cast correctly but cannot prove which way they voted to a third party. This prevents them from selling their vote and then making good on the promise. In addition, voters can vote at any poll site. Unlike previous receipt-free solutions, the framework allows a voter to contest his or her vote and ensure that it was counted correctly. The framework also allows for multiple independent tallying agents.

One audience member asked what the provisions were for a third party, such as a candidate, to contest a user's vote without their consent. The speaker replied that such a party can act as a tallier (because any number of talliers are possible) and tally votes according to their own policy.

WLAN Location Sensing for Security Applications

Algis Rudys

Note: The associated paper is available at *http://www.cs.rice.edu/~arudys/papers/wise2003.html*.

At USENIX '01, someone mounted an active attack and took over the wireless network. People who tried to connect were rerouted to a Web page proclaiming "ALL YOUR WAP BELONG TO US." What if we could track the perpetrator by triangulating signal strength? Thanks to Algis Rudys and his collaborators, now you can!

Their implementation can handle both single and multiple adversaries. The speaker showed a jitter diagram and claimed that even in the presence of multiple adversaries their implementation can get location claims within 3.5 meters. A full version will be presented at WISE 2003.

Trends in DoS Attacks

Jose Nazario

URL: *http://www.monkey.org/~jose/presentations/ddos.d/*

The speaker detailed the preliminary results of an experiment in monitoring Internet traffic for traces of DoS events. Over the past few years, 117K events were monitored using blackhole collection with a view of roughly 1/256 of the Internet. The monitoring view consisted of targets geographically distributed over the world on a Class A network. Overall, there was a 50–50 split between spoofed and real source addresses. The speaker showed graphs of TCP vs. UDP traffic year by year, and noted that there has been a dramatic shift: In 2003 nearly all DoS traffic has been UDP with very little TCP, while the opposite was true in 2002. The presentation also showed a graph of durations of DoS attacks.

One audience member asked if the monitor network was a contiguous Class A network. The answer was yes, contiguous and globally announced.

Another audience member asked about trends in spoofed source addresses; how many are spoofed vs. real? The speaker replied that there was a general trend toward DoS adversaries noticing when they are monitored. As a result, the efficiency of the method was beginning to decrease.

SonicKey

Greg Rose

The speaker demonstrated a system in which a public key, challenge-response, or other data is encoded as a sound playable over a phone or computer speaker. With this system, when you want to send your public key to someone, you just play it for them; their computer or phone receives it via the microphone, decodes it, and obtains your public key. A similar mechanism can be used to allow the phone to answer challenge-response queries from a PC equipped with speaker and microphone. The speaker is employed by Qualcomm, a major cellular phone software company – the SonicKey system is in use at the Qualcomm head office for access control to the building.

One audience member asked whether transaction details were necessary before creating a response. The speaker replied that the system could be deployed just like SecureID.

Another audience member asked about the range of the system. The speaker replied that it was spread spectrum, with a range of 8–9 feet.

Someone asked whether there was dedicated hardware in the phone. The speaker replied that the prototype platform phone does have some dedicated hardware for speeding up RSA public-key crypto operations, but not specifically for the encoding itself.

Another audience member asked whether you can do decoding in the handset. The speaker replied that this was for the next phone generation.

Rekeying

David Molnar

Slides: *http://www.cs.berkeley.edu/~dmolnar/tiny-rekey-wip.ppt*

The speaker gave a short overview of the Berkeley mote sensor network platform and the TinySEC mechanism for link layer authentication and encryption in mica2. The main problem with TinySEC is that keys are baked into motes at code image flash time and cannot be changed afterwards. The speaker presented a protocol for rekeying these sensor nodes and outlined its applications in implementing "secure transient associations."

Honeyfarm

Nicholas C. Weaver

URL: *http://www.cs.berkeley.edu/~nweaver/*

The speaker addressed the question of automatic detection of fast worms. After seeing Sapphire, Slammer, and other quick-spreading worms, we know that it really is possible to own the Internet in 15 minutes. What can we do about this? The speaker introduced a "honeyfarm" as a method for automatic detection of quick-spreading worms. In a honeyfarm, multiple machines running honeyd can act as targets for quick worms and provide researchers with early warning and analysis traces. The speaker also outlined the idea of "wormholes," or network segments designed to draw and contain worm activity. The goal of such a system is to give researchers early warning and, more important, provide a platform for automatic response to new worms.

One audience member asked how many distinct physical machines were in the honeyfarm. The speaker replied that he had about 1000 endpoints in the current deployment.

Honeyd

Niels Provos

http://www.citi.umich.edu/u/provos/honeyd/

The speaker gave an update on the honeyd virtual network host daemon. A machine running honeyd looks like multiple hosts on a network; adversaries probing a network will attempt to attack these hosts, leading to valuable analysis traces. More work is needed, so potential users and volunteers should check out the honeyd page.

DoS Through Regexps

Scott Crosby

URL: *http://www.cs.rice.edu/~scrosby/hash/slides/USENIX-RegexpWIP.2.ppt*

The speaker introduced the idea of causing denial of service by forcing applications to take a long time matching inputs to regular expressions; this is an extension of the idea presented by the speaker and others in a regular conference paper on "Algorithmic Denial of Service Attacks." The key here is that instead of converting regexps to Nondeterministic Finite Automata (NFA), determinizing to form Deterministic Finite Automata (DFA), and then running, many implementations simply attempt to match regexps nondeterministically directly. In naive implementations, this can lead to exponential time. Perl has optimizations that claim to limit the running time, but it's not clear whether these are correct.

The speaker then gave a simple example of a regexp that could take exponential time, namely a*[ab]*O, and walked through the process by which the regexp parsing could take this much time. Then the speaker gave an example of a regexp "in the wild" that could take exponential time in the worst case – this one from SpamAssassin. The key issue was the number of "." in the regexp. In theory a spammer could use this to cause a DoS attack on individuals using SpamAssassin.

Someone asked if you could avoid the problem by converting the regexp to an NFA and parsing from the equivalent DFA. The speaker noted that the conversion to DFA introduces an exponential blowup in the size of the state machine.

Readers who want more background on NFAs and DFAs may want to look at Lewis and Papadimitriou's *Elements of the Theory of Computation* for a theoretical perspective. For a practical perspective on regular expressions and their matching, readers may wish to consult Freidl's O'Reilly book *Mastering Regular Expressions*.

Porting Trusted BSD to Darwin

The speaker discussed Apple's approach to migrating mandatory access control and other security features from recent BSDs to Darwin, and the challenges involved. These features have also appeared in systems such as SELinux and Flask/TE. The presentation provided a short overview of the Darwin architecture: Darwin = NextStep, recent Mach (monolithic Mach), parts of FreeBSD 3 and 4, and IOKit. The speaker then went over new security features added to BSD since the creation of Darwin, such as sbuf. One of the key issues involved is the treatment of IPC and threaded processes; Darwin manages a mapping between BSD and Mach notions of processes, and some of the work requires working around the Mach IPC model.

Still Cleartext After All These Years

This talk was a follow-up to Dug Song's presentation at the USENIX Security WiP session in 2001. At that talk, Song showed that many conference attendees were divulging passwords over the open 802.11b network. The presenter asked the question, "Have we learned anything since then?" After all, USENIX Security is supposed to be a gathering of the best of the best security experts. Are we practicing what we preach in security? The speaker mounted active and passive attacks on the (open, unencrypted) conference wireless network. Tools used included dsniff, honeyd, and netics. In addition, the speaker worked with Niels Provos to create an "SSH mirage" – a fake AP that trapped SSH connections and attempted to mount a man-in-the-middle attack on unwary conference goers.

After discussing the methods, the speaker posted a slide full of passwords. "If you see your password up here, you might want to change it." No usernames were provided at that time, to allow victims to change their passwords. Unencrypted Web authentication provided many of the passwords, as well as AOL Instant Messenger and other IM clients.

In a point of good news, no Telnet traffic was observed, only SSH. In addition, only one conference attendee fell for the SSH mirage; the others heeded the warning "HOST KEY HAS CHANGED" and thought better of it (or maybe people just didn't get close enough to the fake AP?). The speaker's conclusion was that many people, even at USENIX Security, are not yet taking wireless security seriously enough, even though there are some marginal improvements over 2001.

At the end of the talk, one audience member pointed out that encrypted instant messaging is available via Jabber or the Trillian AIM client.

USENIX Upcoming Events

The 6th NordU/USENIX Conference (NordU04)

Co-sponsored by EurOpen.SE and USENIX

JANUARY 28–FEBRUARY 1, 2004, COPENHAGEN, DENMARK
http://www.nordu.org/NordU2004/

Asia BSDCon 2004

Co-sponsored by USENIX

MARCH 12–15, 2004, TAIPEI, TAIWAN
http://www.asiabsdcon.org/

2004 USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)

Co-sponsored by USENIX, ACM SIGCOMM, and ACM SIGOPS

MARCH 29–31, 2004, SAN FRANCISCO, CALIFORNIA
http://www.usenix.org/events/nsdi04

Third USENIX Conference on File and Storage Technologies (FAST '04)

In Cooperation with ACM SIGOPS, IEEE Mass Storage Systems Technical Committee (MSSTC), and IEEE TCOS

MARCH 31–APRIL 2, 2004, SAN FRANCISCO, CALIFORNIA
http://www.usenix.org/events/fast04
Work-in-Progress (WiPs) proposals due: March 19, 2004

Third Virtual Machine Research and Technology Symposium (VM '04)

Sponsored by USENIX in cooperation with ACM SIGPLAN

MAY 6–7, 2004, SAN JOSE, CA, USA
http://www.usenix.org/events/vm04

The Second International Conference on Mobile Systems, Applications, and Services (MobiSys 2004)

Jointly sponsored by ACM SIGMOBILE and USENIX in cooperation with ACM SIGOPS

JUNE 6–9, 2004, BOSTON, MA, USA
http://www.sigmobile.org/mobisys/2004/

2004 USENIX Annual Technical Conference

JUNE 27–JULY 2, 2004, BOSTON, MA, USA
http://www.usenix.org/events/usenix04
Paper submissions due: December 16, 2004

2004 Linux Kernel Developers Summit

JULY 18–19, 2004, OTTAWA, ONTARIO, CANADA

13th USENIX Security Symposium

AUGUST 9–14, 2004, SAN DIEGO, CA, USA
http://www.usenix.org/events/sec04
Paper submissions due: January 25, 2004

The 4th International System Administration and Network Engineering Conference (SANE 2004)

SEPT. 27–OCT. 1, 2004, AMSTERDAM, THE NETHERLANDS
http://www.sane.nl

18th Large Installation Systems Administration Conference (LISA '04)

NOVEMBER 14–19, 2004, ATLANTA, GA, USA
http://www.usenix.org/events/lisa04/
Extended abstracts and invited talk and workshop proposals due: April 20, 2004

Sixth Symposium on Operating Systems Design and Implementation (OSDI '04)

DECEMBER 6–8, 2004, SAN FRANCISCO, CA, USA
http://www.usenix.org/events/osdi04
Paper submissions due: May 26, 2004

For a complete list of all USENIX & USENIX co-sponsored events, see
http://www.usenix.org/events

CONNECT WITH USENIX

MEMBERSHIP AND PUBLICATIONS

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Phone: 1+ 510 528 8649
FAX: 1+ 510 548 5738
Email: office@usenix.org
login@usenix.org
conference@usenix.org

WEB SITE

http://www.usenix.org

EMAIL

login@usenix.org

COMMENTS? SUGGESTIONS?

send email to ah@usenix.org

CONTRIBUTIONS SOLICITED

You are encouraged to contribute articles, book reviews, photographs, cartoons, and announcements to *;login:*. Send them via email to *login@usenix.org* or through the postal system to the Association office.

The Association reserves the right to edit submitted material. Any reproduction of this magazine in its entirety or in part requires the permission of the Association and the author(s).

USENIX

The Advanced Computing Systems Association

;login:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES