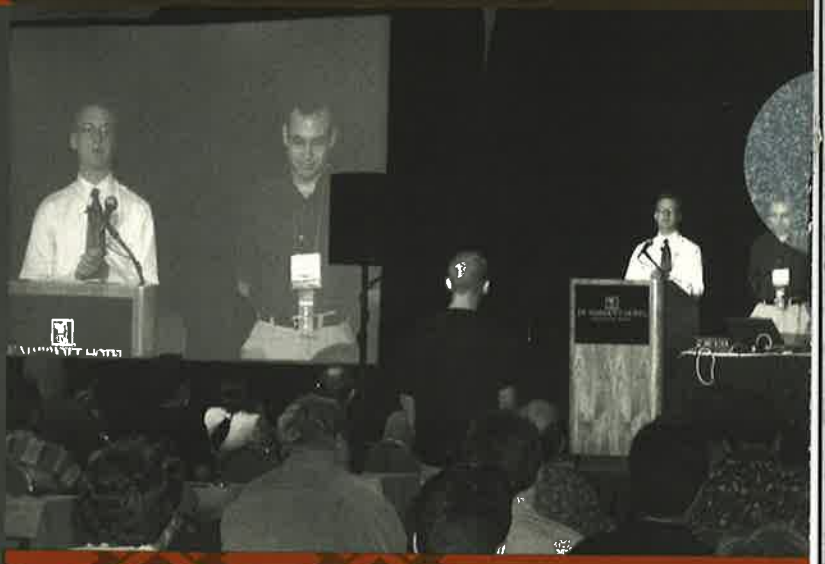
;login:
THE MAGAZINE OF USENIX & SAGE
November 2001 • Volume 26 • Number 7

Special Focus Issue: Security
Guest Editor: Rik Farrow



inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

USENIX Upcoming Events

15th Systems Administration Conference (LISA 2001)

Sponsored by USENIX & SAGE

DECEMBER 2–7, 2001 San Diego, California, USA

http://www.usenix.org/events/lisa2001

Conference on File and Storage Technologies (FAST)

Sponsored by USENIX, in cooperation with ACM SIGOPS and IEEE TCOS

JANUARY 28–30, 2002 Monterey, California, USA

http://www.usenix.org/events/fast/

BSDCon 2002

FEBRUARY 11–14, 2002 San Francisco, California, USA

http://www.usenix.org/events/bsdcon02/

Camera-ready final papers due: December 4, 2001

The 4th NordU/USENIX Conference (NordU/USENIX 2002)

Co-sponsored by USENIX and EurOpen.SE – The Swedish Association of UNIX Users

FEBRUARY 18–22, 2002 Helsinki, Finland

http://www.nordu.org/NordU2002

Final papers due: December 7, 2001

The 3rd International SANE Conference

Organized by NLUUG

Co-sponsored by USENIX and the NLnet Foundation.

MAY 27–31, 2002 Maastricht, The Netherlands

http://www.sane.nl

2002 USENIX Annual Technical Conference

JUNE 9–14, 2002

Monterey, California, USA

http://www.usenix.org/events/usenix02/

2nd Java™ Virtual Machine Research and Technology Symposium (JVM '02)

AUGUST 1–2, 2002 San Francisco, California, USA

http://www.usenix.org/events/jvm02

Paper submissions due: February 4, 2002

Notification of acceptance: March 12, 2002

Camera-ready final papers due: May 28, 2002

Registration materials available: April, 2002

11th USENIX Security Symposium

AUGUST 5–9, 2002 San Francisco, California, USA

http://www.usenix.org/events/sec02

Paper submissions due: January 28, 2002

16th Systems Administration Conference (LISA '02)

Sponsored by USENIX & SAGE

NOVEMBER 3–8, 2002 Philadelphia, Pennsylvania, USA

2nd Workshop on Industrial Experiences with Systems Software (WIESS '02)

Sponsored by USENIX

Co-sponsored by ACM SIGOPS & IEEE TCOS

DECEMBER 8, 2002 Boston, Massachusetts, USA

http://www.usenix.org/events/wiess02

Submissions due: July 15, 2002

5th Symposium on Operating Systems Design and Implementation (OSDI)

Sponsored by USENIX

Co-sponsored by ACM SIGOPS, & IEEE TCOS

DECEMBER 9–11, 2002 Boston, Massachusetts, USA

http://www.usenix.org/events/osdi02

Submissions due: May 24, 2002

contents

;login: Vol. 26 #7, November 2001

;login: is the official magazine of the USENIX Association and SAGE.

;login: (ISSN 1044-6397) is published bimonthly, plus July and November, by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

$50 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $60 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 photo: SDMI Q&A: Scott A. Craver, presenter, and Dan Wallach, Program Chair (see page 7)

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

This has been a summer of disasters. Natural ones, like hurricanes, droughts, and tornados. National ones, most incredibly the suicide bombings. The dot-com revolution has become the dot-bomb devolution. And for security, an embarassment of automated attacks against IIS servers, making a travesty of claims for security.

While some things, like nature, fanatics, and the stock market, are not under anyone's control, other things are. Computer and network security relies most on simplicity – if a service is not run, it can't be exploited. When you read this issue of *;login:*, please keep this in mind. It should be easy, because you will find this being said more than once.

I am very excited about this issue. More people contributed articles and summaries than we could fit into our pages. For those of you who could not attend the Tenth USENIX Security Symposium this August in D.C., I recommend the summaries. Reading them is the next best thing to having been there. We had more people volunteer to write summaries than we had sessions to summarize, and I thank all who contributed their time and energy in taking notes and writing them up for us.

The articles have been divided into three groups: Forensics, Intrusion Detection, and Best Practices. Keith Jones, of Foundstone, contributed two articles based on his experience working on live systems with hacking tools installed. I appreciated reading both of these articles, but especially liked Keith's description of knark, and techniques for defending against tools like it, as well as an excellent description of how loadable kernel modules work as almost undetectable rootkits.

Brad Powell, of Sun Microsystems, writes about another aspect of forensics, detecting trojaned commands. In some ways, what Brad describes are simple tools for collecting checksums using MD5. What he barely touches on is the amount of work that went into this project, which includes a database of MD5 checksums – one for every binary that Sun has shipped. During a Sun BoF at the conference, I heard that Caspar Dik had spent many hours loading all the Sun distribution CDs into drives so that all distributions and patches could be added to the database. I would love to see all vendors provide both a database like Sun's, as well as an easy-to-use interface for checking the authenticity of files. I also want to thank Brad, and his companions at Sun, for actually completing something that many of us have wished for.

In the Intrusion Detection section, we have three articles from people who have implemented ID systems using open source software. Jon Lasser talks about using snort in an environment where, even with carefully trimmed rulesets, he was seeing tens of thousands alerts each day. Peter Van Epp writes about the uses of Argus, a tool for recording IP header information. Peter talks enthusiastically about many of the practical uses he has found for the gigabytes of Argus data that he has collected. In addition Sven Dietrich covers a different angle of IDS: survivability, which is the ability of a distributed ID system to continue running in the face of determined adversaries, even when the adversaries are insiders.

The final section borrows from the experience of three gentlemen who work in the open environments of universities and research institutes. Oscar Bonilla, of Galileo University in Guatemala, describes how necessity forced him to design a firewall that fits on a floppy disk. Remember what I wrote earlier, about not running unnecessary

services? Oscar shows you how to do this in the extreme case of a practically diskless server.

David Brumley, of Stanford University, and Abe Singer, of the San Diego Supercomputing Center, describe the best practices that have made their sites more secure and more easy to manage. David focuses on several very practical issues, with simplifying configurations high on the list. Abe describes how the San Diego Supercomputer Center has terminated the use of plaintext passwords in network services entering SDSC. Both explain their successes and failures, and extol the power of having management buy-in when it comes to getting anything done. I especially liked the psychological techniques that Abe suggests for getting troublesome researchers to migrate to new, more secure, tools and protocols.

Due to length restrictions, articles by Erin Kenneally, Paco Hope, and Ofir Arkin will appear in later issues of *;login:*. I truly wish we could have fit everything into this one issue, but we ran out of available pages.

As you read the summaries, think about the implications touched on by the special evening session about the SDMI challenge.

A group of researchers decided to investigate proposed techniques for protecting some intellectual property – in this case, music owned by corporations and distributed on CDs. The researchers were successful in reverse engineering and countering most, perhaps all, of the six techniques used to watermark music or protect the contents track. But when it came time to present their paper, they were threatened with a lawsuit, based on the Digital Millenium Copyright Act (DMCA). It was only with the support of USENIX and the Electronic Freedom Foundation that they were able to present the results of their research.

As in the case of Slylarov, who was arrested after explaining the extremely lame job Adobe had done to protect copyrighted information used in their e-Books (XOR-ing a constant against the bytestream), the DMCA seeks to prevent researchers from pointing out that the emperor has no clothes. Imagine a world where you can only report dangerous defects in automobiles to the manufacturer. This is the world that DMCA expouses, where it is illegal to discuss mechanisms used to protect copyrights. (See *http://lwn.net/2001/0726/bigpage.php3* regarding the Skylarov case).

In the wake of the attacks on the World Trade Center and the Pentagon, the same old voices are crying out for new limitations on encryption, and less restrictions on the right to search, to incarcerate, and generally, to trample on the US Bill of Rights. This trend toward a corporate state – one that focuses on corporate rights instead of citizen rights – began before the suicide bombings.

I would much rather live in a world made secure through correctly designed software and encryption without backdoors, than in a police state where it is illegal to share a thought privately, disagree with the government, or even test whether the software that is supposed to be secure is really doing anything at all. We are living in a most frightening time, and it is not only terrorists I am frightened of.

conference reports

This issue's reports are on the 10th USENIX Security Symposium

Our thanks to the Summarizers:

Takeaki Chijiiwa
Sameh Elnikety
Kevin Fu
Rachel Greenstadt
Yong Guan
Anca Ivan
George M. Jones
Stefan Kelm
David Richard Larochelle
Ross Oliver
Evan Sarmiento
Cole Tucker
Mike Vernal
Sam Weiler

10th USENIX Security Symposium

Washington, DC
August 13–17, 2001

KEYNOTE

Web-Enabled Gadgets: Can We Trust Them?

Richard M. Smith, CTO, The Privacy Foundation

Summarized by George M. Jones

Richard Smith started off by saying that what he primarily does is "cause problems," mostly for companies that have not thought through the security implications of products that they have released. They often "discover unintended consequences that companies don't like to talk about." The three main areas they consider are security, privacy, and control.

Richard M. Smith

He stated that "consumers care more about the security of cell phones than about Web servers" because cell phones are personal devices with which consumers have immediate connections. Application developers and companies are more concerned with functionality than security. Products such as consumer devices based on real-time operating systems tend to have lower concerns for security.

Smith said that DirecTV was the first consumer device that got his interest about privacy issues. It had a phone jack. What information was it sending back? Later, a call to customer service on a different issue revealed that the customer service people were able to send commands (via satellite?) to turn his TV on. Is this a good thing? Still another time the company apparently chose to "advertise" new services by causing the TV to tune to a soft-porn channel which he had not subscribed to or selected.

Earlier this year, just before the Super Bowl, the company downloaded a program to all DirecTV boxes. The goal was to disable black market devices used to pirate programming. It succeeded. But what if they had made a mistake? What if they had disabled service for legitimate customers? Who, in fact, owns the boxes? DirecTV clearly did not own the black-market devices. Did the company's actions constitute "hacking"? Did the terms of service allow them to reprogram the legitimate boxes?

It turns out that DirecTV was not sending back "Nielsen" information, just a lot of information about the temperature inside the box. Their competitor Tivo does send in "Nielsen" info. You have to explicitly opt out by calling customer service.

We're entering a brave new world of connected devices. A company called Sports Barn sold a strap-on device that monitored your daily exercise...and then uploaded it via phone to their Web site to create a "personal profile" (which, of course, would never be used for marketing or other) purposes. One could have gotten the same effect by uploading to a PC without disclosing personal information, and there are inexpensive stand-alone devices available at sports shops that do similar things. But to maintain a record you might have to (gasp) write things on paper: the price of privacy. Oh, the company just went out of business. Many formerly happy customers now have worthless devices. Similar things could never happen with subscription software licensing, could it? Software companies never go out of business, get bought out, refocus on newer products, or have turnover or loss of support staff.

Ever considered plugging your picture frames into the phone? Kodak wants you to so that you can "register" your digital pictures. Of course, you'll pay a recurring subscription fee to do so. And they'll never share your private pictures with anyone either, their servers never get hacked, and all their employees are intimately familiar with their information security policies and actively make it their top priority each day to follow them.

Want free wireless Internet access? See the Global Access Wireless Database at *http://www.shmoo.com/gawd/*. Want to see what your neighbors and coworkers are doing? 802.11 is your friend. At least one non-USENIX conference person was observed using the USENIX wireless network at the symposium.

Convergence is a good thing, right? Fewer devices, more functionality, lower cost, but do you really want someone using the cell phone API in your combo phone/palm pilot to run a program that (1) turns off the speaker, (2) places a call, and (3) turns on the microphone? Your phone is now a bugging device, in addition to a tool for pinpointing your location at all times. Personally, I'll stick with dumb one-way pagers and only turn my phone on when I want to make a call (and announce my location).

Do you ever store personal/low-sensitivity data and business/high-sensitivity information on, say, a palm pilot, a laptop computer, or a home computer connected to public networks? Mudge and Kingpin of @stake pointed out in a later talk (dressed in bathrobes to protest their 9 a.m. speaking slot) that PalmOS has serious security problems. Cable modem providers do not generally provide security/firewall services. Laptops are routinely stolen (the laptop that was being used as the gateway/router for the conference terminal room disappeared overnight and one of the terminal-room attendants stopped someone who attempted to walk out with its replacement in broad daylight). Information security management issues that were once handled by trained security professionals in controlled, centralized environments are now the problem of your grandmother, Joe Six-pack, and your CEO.

Do you ever speed in a rental car? In the Q&A session Rik Farrow noted that at least one person has been fined by the rental car company for doing so. The company installed GPS devices in all it's cars that enable it to track down stolen cars...and tell how fast you go. Any guesses how long it will be before law enforcement and insurance companies push for legislation requiring such devices in all new cars?

Steve Bellovin noted that during the talk there had been multiple nmaps of the wireless net and an ongoing battle for address of the default router (Dug Song of dsniff fame was in the room).

And lastly, true confessions — Smith admitted that he has not turned on WEP on his own wireless net at home.

And the beat goes on...

Invited Talks

A Maze of Twisty Little Statutes, All Alike: The Electronic Communications Privacy Act of 1986 (and Its Application to Network Service Providers)

Mark Eckenwiler, U.S. Department of Justice

Summarized by Cole Tucker

The Electronic Communications Privacy Act of 1986 has a reputation for complexity. Mark Eckenwiler gave an expressive overview of the law, primarily from the viewpoint of a system administrator/provider. Basically, the act covers the relationship between, providers and customers, and providers and the government. It tries to allow for communication privacy while keeping in mind that online records are the key to prosecuting network criminals.

The law distinguishes four types of environments, based on whether the data is content or transactional in nature and whether it's being intercepted in real time or after it's been stored. The class that receives the most protection, real-time content, has a very basic rule for the normal user: don't get or look at it. For the government, the rule is nearly as simple: don't get it without a wiretap order. Providers aren't supposed to look at it unless they're in the process of protecting their rights and property. So if you're a regular user, don't run an unauthorized sniffer. If you're representing a provider, under Eckenwiler's interpretation, feel free to run an IDS or even a keystroke logger in real time; you can be proactive in defending yourself. Other exceptions are made for publicly accessible systems, such as IRC, or if all parties consent, say in a system that has a banner stating that use implies consent to monitoring. As a provider, if you have a legitimate need to monitor, there's no reason to worry.

The second class of data consists of transactional records being intercepted in real time. For providers and users the rules remain nearly the same: hands off for the latter and have a good reason for the former. The standards have been lowered for the government, so this information is essentially "less private." For access to this data, the government simply needs a court order. Examples of data that fall under this are addresses attached to incoming emails and information on where users are connecting from and whether they are online.

Next comes stored content. Eckenwiler referred to this section as "Dichotomies 'R' Us"; basically, each situation has different rules that apply, with way too many to generalize here.

Finally, there are stored transactional records. Users, hands off. Providers are

allowed to reveal this information to anyone they like, except for the government. In respect to the government, there are two classes of data: basic user data and non-contact info. Basic user data (things like name, address, and phone number) is accessible with a subpoena, and thus not strongly protected. Everything else requires a 2703(d) warrant to access, but providers can be sent a court order requiring they hold on to the data for a specified amount of time, usually in expectation of a warrant being served in the near future.

Loaning Your Soul to the Devil: Influencing Policy without Selling Out

Matt Blaze, AT&T Labs-Research

Summarized by George M. Jones

Matt Blaze commented on the public debate over cryptology that's taken place over the past 10 years or so. He included amusing stories of "hacker tourism," including nine cryptography experts all independently trying to score "cool" points by stealing stationery from secret congressional briefing rooms and NOT opening a red folder marked "TOP SECRET: President's Daily National Security Briefing" when left alone in a conference room in the old executive office building.

What can a scientist/techie contribute to the public policy debate? His main advice is "stick to what you know" (science/technology). "You are listened to because people believe you have objectivity.The basic purpose of science and engineering is to expand understanding of reality/truth, with no compromises." You are not there to comment on philosophy, politics, or constitutional law.

He gave an amazingly insightful list of the contrasting values of science and politics):

- Science is interested in finding truth. Politics is about balancing interests.
- In science, people are rewarded for new discoveries. Disruptiveness is considered good. In politics, people are rewarded for making other people happy. Disruptiveness is considered bad.
- In science, uncompromising people are admired; in politics, uncompromising people are considered fools.
- In science, "honesty" means admitting mistakes; in politics, it means keeping promises.
- In science, challenging someone shows interest; in politics, a challenge is an attack.
- In science there is no "dress code"; in politics, even suits can be considered "casual" (and thus cause you not to be taken seriously).

The policy options range from discouraging/forbidding its use; allowing limited strength crypto; allowing use of strong, modern cryptographic methods; and encouraging use. In the last few years the US has moved mostly from the first to the third stage.

Matt Blaze

The tone of the debate has also changed and includes more actual dialogue. We no longer have one side yelling, "You're a bunch of long-haired hippies," and the other yelling, "You're a bunch of jack-booted thugs." Now it's just "you're a bunch of hippies" vs. "you're a bunch of thugs." See, for instance, "Thou shalt use skipjack/clipper" vs. the process for selecting AES.

"Washington, D.C. is another planet, a closed system." "Much of what happens here is for show." "Any meeting with a policy maker involves a little conspiracy to make each other feel important." "Meetings with congressional staffers always end with the question 'What do you suggest we do?'" Stick to what you know.

Cops Are from Mars, Sysadmins Are from Pluto: Dealing with Law Enforcement

Tom Perrine, San Diego Supercomputer Center

Summarized by Ross Oliver

Tom Perrine described some of his experiences with law enforcement people and discussed his recommendations for other sysadmins who may need to interact with law enforcement.

Like system administration, law enforcement is a culture as well as an occupation, with its own lingo, inside jokes, etc. There are also many different law enforcement agencies: federal, state, city, county, military, and customs. Even schools and universities often have their own police force.

Throughout the talk, Perrine emphasized the importance of trust in individuals rather than organizations. Just as in any large organization, there are "clueful" and not "clueful" members, and building personal relationships is key. Also realize that the goals and priorities of law enforcement may be different from yours.

Because they are "agents of the government," law enforcement officers have many legal constraints on their actions that may not apply to private citizens. Sysadmins can take advantage of "ISP exemptions" in the law to take "any steps necessary to protect the communications system."

Perrine recommends that sysadmins become familiar with applicable laws (both federal and state) before the need to apply them arises. Advice of qualified legal counsel is strongly recommended. Also, make sure your organization's policies are suitable, and adhere to them during any investigation.

READING BETWEEN THE LINES: LESSONS FROM THE SDMI CHALLENGE

Summarized by Rachel Greenstadt

Scott A. Craver, Min Wu, and Bede Liu, Princeton University; Adam Stubblefield, Ben Swartzlander, and Dan S. Wallach, Rice University; Drew Dean and Edward W. Felten, Princeton University

Program Chair Dan Wallach introduced this talk as being a long time in the making and mentioned that he was pleased to have it here. However, he stressed that this first section would be a normal, boring technical talk. THEN there would be a panel discussion where policy questions would be allowed. Matt Blaze asked when the subpoenas would be served; however, despite the large mass of press and lawyers that joined the USENIX attendees, there was no last-minute withdrawal of the talk this time, and no FBI agents came to cart Scott Craver away as he gave his talk.

Craver began by describing the challenge, which took place during three weeks in September and October of 2000. SDMI (Secure Data Music Initiative) invited "hackers," otherwise known as the general public, to crack six of their proposed technologies labeled A through F. There were four watermarking technologies and two others. SDMI offered a cash prize for the successful defeat of one of their technologies, but this required the winners to sign a Non-Disclosure Agreement, so the Felten group decided to forego the prize in favor of publishing their findings.

SDMI is an organization, an initiative, and the technology for that initiative. At the time of the challenge, that technology was watermarking and related technologies. The watermarks (technologies A, B, C, and F) were composed of a robust and a fragile component, the robust part of which would survive altered music. Through a missing watermark in the fragile component, such as if it had been mp3 compressed, an SDMI device could perhaps determine if a CD track was ever an mp3 in the past, perhaps illegally downloaded. The other technologies (D and E) were used to sign tables of contents, supposedly to control the propagation of CDs with mixed tracks.

For the watermarking technologies there were three samples given: (1) a sound clip without a watermark, (2) the same sample with a watermark, and (3) a different sound clip with a watermark. The challenge was to remove the watermark from the third sound clip. SDMI provided no actual embedders or detectors. There was an online oracle to which you could submit a sound clip and get a response. There was no description of the algorithms used, and no details or reasons were given when an oracle rejected a clip. The challenge lasted only three weeks and the oracle had a turnaround time in hours. As such, adaptive oracle attacks, which would be possible if the system were deployed, were not feasible.

There were several approaches used against the marks: (1) brute force attacks not specific to the algorithm used and which mostly consisted of adding noise and filtering, (2) slight brute force attacks loosely based on supposed details of the algorithms, and (3) full-blown reverse engineering.

For technologies B and C, the group noticed that there was a narrow band signal added to the clip. By the slightly brute method of filtering at the frequency and adding narrow-band noise they were able to foil the oracle.

In their analysis of technology A, the group noticed a slight warping in the time domain as though the signal was slowly advancing or decreasing. They determined that this phase shifting was pre-processing and not the actual watermark, since the oracle did not admit the sample when the distortion was removed. However, removing this distortion in technology F was able to make that watermark undetectable (quick, somebody call the FBI).

Another approach to defeating technology A would have been to try reinstating the fragile component. However, there was no way to test this type of attack using the oracle.

The group noticed a ripple in the frequency domain, which led them to believe that technology A used some sort of echo hiding technique consisting of deliberate but inaudible echoes, which meant that there was a signal which was delayed and then added back into the music. They tried a filtering approach to reduce the audibility of the echo sufficient to remove the watermark. Wanting to discover more, they decided to do a patent search figuring correctly that this was a proprietary algorithm with a patent. They found a patent belonging to Aris corporation which became Verance, one of the SDMI companies. This made them feel like they were on the right track. They also discovered that it was a simple echo every fiftieth of a second and that a delayed version was added or subtracted every fifteenth interval. To further analyze the signal they used the auto-kepstral technique for echo hiding, combining techniques to estimate the echo. They've come up with better echo hiding detection software subsequent to the challenge. Scott demonstrated a program that was color coded to detect the echo.

For technologies D and E, SDMI presented table-of-contents files for 100 CDs and signature tracks. The challenge was to create a new table of contents and successfully forge a signature for it. For technology D they found that all the energy was concentrated in a small frequency band of 80 frequency bins which only actually used a 16-bit signal

repeated five times with constant shuffling. Since there were only 16 bits of output, a user should be able to acquire many authenticators, as there were two hash collisions among the CDs given. However, it was difficult to get further than this analysis because the oracle for D didn't work; it would always return "invalid" regardless of input. Technology

Q & A: Scott Craver & Dan Wallach

E, however, didn't have any data to analyze at all. You could submit a mail saying you'd try mixing this track and that track, and you'd get a reply saying that you couldn't do that.

The speaker concluded by saying that many claimed that this was a system to "keep honest people honest." However, though the Felten group felt that the system was too complex for that, they wouldn't claim any type of strong security. The systems require trusted clients in a hostile environment, but if deployed they would be broken quickly. No special EECS knowledge is needed and there are no dirty secrets. Anyone with reasonable expertise could do this. Watermarking can be useful but not in this situation. The weakness is in the overall concept, not the specific technology. One main lesson learned is that security through obscurity STILL doesn't work. This is particularly the case for secret algorithms which are patented and therefore public.

Peter Honeyman asked about the possibility of a secure watermark. Scott replied that he personally thinks that watermarking won't work for actively enforcing a usage policy since doing this provides all targets an oracle that they can use. He is pessimistic about the use of watermarking for copyright control. He clarified that they broke, according to the oracle, technologies A, B, C, and F, but that D and E had no valid responses. He also clarified that only technology A used echo hiding, and that though they don't know what the criteria for the oracle was, it appeared to make a decision based on detectability and quality. He explained that some areas where watermarking might prove useful is in fragile watermarks which provide tamper evidence in digital photographs and in preventing duplication of currency. These technologies have a different threat model. Someone asked about copy protected CDs; Scott replied that that was a completely different approach done entirely at the hardware level. People wondered why honest people would not want a complex copy protection scheme; Scott answered that complex schemes have higher rates of failure and higher cost. Someone asked how this was relevant to detecting steganographic information and Scott answered that they were basically the same and that the information about echo detection would be useful.

Panel Discussion on SDMI/DMCA

Moderator: Dan Wallach, Rice University; Panelists: Edward W. Felten, Princeton University; Cindy Cohn, EFF; and Peter Jaszi, American University College of Law

The three panelists spoke about the legal and social questions surrounding the SDMI/DMCA issue. Dan Wallach mentioned that if there were any representatives from the record company, the panel would love to have someone from the other side come speak; he doubted, however, that they would be here.

Peter Jaszi then presented a detailed description of the Digital Millennium Copyright Act (DMCA), section 1201. He explained the difference between the DMCA and copyright law. Copyright law has been developed and refined over a few hundred years and maintains a delicate balance between owners and users' privileges. To that end it has been relatively successful. It is important to understand that the DMCA is not copyright law but, rather, a supplement to copyright law or para-copyright legislation. As such it has the potential to override the copyright default protections which had been carefully laid out over time. He sought to explain these overrides and mentioned that the risk the DMCA poses to the fundamental copyright system isn't news and wasn't news when it was passed. As a result some limitations to the DMCA were built in, but most of these exceptions are not very functional.

Prof. Peter Jaszi

The fundamental commandment of the DMCA is "Thou shall not circumvent for access." The fact that it was access and not use was a compromise intended to limit the DMCA. However, it limited the legislation less than some imagined it would since there is a great deal of confusion between access and use. There are also secondary prohibitions concerning making goods and services which can be used for circumvention available. Section 1201(b)(1) can be interpreted broadly, and it was under this provision that the threats from SDMI to the authors of the paper were made.

Section 1201(c) presents a fair-use exception in wonderful ringing language, however, it is completely irrelevant since it references fair use as a defense of copyright and the DMCA is not copyright.

The law enforcement exception is actually sweeping and robust; it applies to all the provisions of the act. The reverse engineering exception is not half bad; it refers to the whole range of prohibitions although it is still narrower in scope than the protections under copyright law. Sections 1201(g) and (j) present limited exceptions for encryption research and security testing which are uncertain in scope. Section 1201(h) presents a small but robust exception to allow adults to circumvent in order to frustrate a minor's attempt to achieve privacy in a Web environment. Section 1201(i) allows ordinary people to protect their privacy, but it is only a conduct exception; you need to make your own tools and not distribute them. There is less to all these limitations and exceptions than meets the eye.

There are risks posed by this legislation to the traditional balance of interest in copyright law, which calls for a pushback against legislative excess. To this end Jaszi is forming a new access coalition. They have a Web site at *http://www.ipclinic.org*.

Cindy Cohn from the EFF said that Peter had already said everything about section 1201 but stressed that the EFF was "pushback central" and explained ways in which people could get involved in this effort. The EFF has been involved in this issue even before the cases involving *2600 Magazine*, Felten and the USENIX presentation of the SDMI paper, and the California trade secrets case.

Thomas Greene from the *Register* wondered why the mainstream press hasn't realized their stake in this and what it implies about freedom of the press. Cindy replied that they were getting increased press support with the Skylarov arrest; speaking speculatively, she also mentioned that the mainstream press is owned by content holders. In addition, most press organizations wish to be seen as nonpartisan and objective.

Someone asked about dual use technologies, such as echo detection. The DMCA takes this into account but the language doesn't give much comfort. There are a series of criteria which will give you liability.

There was a question about potential connections between the lawsuit and the Skylarov case. Cindy answered that in strictly legal terms there was no overlap.

Someone asked what to do in Felten's situation, what lessons had they learned? Felten responded that they learned a great deal responding to the threats regarding the paper. He said to talk to people who've been there and keep in mind your goals and values.

Someone brought up the question of whether a person would be at risk for summarizing the session. Cindy said that the letter they received only pertained to the particular paper, but because the paper can be published, prosecuting for summarizing would be hard. Peter suggested that if you were going to synthesize the talk (uh...this is starting to sound disturbingly familiar...) and discuss strengths and weaknesses, theoretically you could be in trouble. Especially if you implement something based on the presentation. Felten mentioned that the fact that this question has no simple answer is telling.

Someone suggested widespread civil disobedience as the only way to effect change. Cindy responded that she never advises people to break the law. Though she feels that if the law is out of step with what people believe their rights are, the law should be changed. Peter added that copyright law has functioned well based on shared social investment. Like the tax code, it works not because it is policed but because there is a high degree of collective buy in to its premises. The most corrosive thing about the DMCA is that the basic assumptions it makes about people are dark and pessimistic. We need to question those assumptions and what flows from them.

Someone asked for some insight into why the industry wouldn't want this research since it would allow them to build better protection schemes. Felten responded that to us the question is, is this technology weak? We didn't make it weak, and we think it should be fixed. The industry's concern is not whether the technology is strong or weak so much as whether people believe it is strong or weak. They think that if the public reaches a consensus that the technology is strong, that will be enough. Many of us find this hard to understand.

[More information and photographs can be found at *http://www.usenix.org/events/sec01/index.html*

Changes in Deployment of Cryptography, and Possible Causes

Eric Murray, SecureDesign

Summarized by Takeaki Chijiiwa

A survey of cryptography deployment was conducted last year (2000) by Eric Murray, and a similar survey was conducted in 2001 to measure changes in the deployment of SSL (Secure Socket Layer) and TLS (Transport Layer Security) Web servers.

The results of the 2000 survey showed 10,381 unique hostname and port number combinations compared to 12,630 in 2001. Detailed results are available at *http://www.lne.com/usenix01*.

There were several noteworthy changes between the results from the surveys in 2000 and 2001:

What got better?

- A 14% increase, 5% decrease, and 8% decrease among servers categorized as Strong, Medium, and Weak, respectively.

- The number of servers supporting 1024-bit key size increased by 10% while a decrease of 8% was seen for support of less than 512-bit key size.
- The protocol adoption saw a shift from SSL v2 (3% decrease) toward TLS (5% increase).

What got worse?

- The number of expired certificates increased from 3.1% to 3.7%.
- Self-signed certificates increased from 0.8% to 2.0%.

The results presented raised many questions from the audience.

Question: Why do you think there was an increase in the number of self-signed certificates?

Answer: This may be due to people playing around with OpenSSL, or the survey may have picked up servers used for internal use. Furthermore, the increase in the number of expired certificates may have been a result of study error and/or the inclusion of abandoned Web sites.

Question: Did you retest the servers from last year's survey?

Answer: No. This was a new list and, therefore, a completely new survey.

Question: Is the raw data available?

Answer: You can email ericm@lne.com for private requests.

Question: Which browsers do you use for personal use?

Answer: Linux and Netscape.

Reversing the Panopticon

Deborah Natsios, cartome.org; John Young, cryptome.org

Summarized by Mike Vernal

Deborah Natsios described the mission of cartome.org and cryptome.org as an attempt to reverse the one-way flow of information controlled by the national surveillance state. Based upon the assumption that information is power, Natsios likened the work of cartome.org and cryptome.org to that of Ariadne in the myth of Theseus and the Minotaur. By reversing the flow of information, cartome.org and cryptome.org hope to empower those who may be caught in the labyrinth of the security state, much as Ariadne empowered Theseus with a trail of silk thread through the labyrinth of Crete.

John Young continued by explaining that cryptome.org welcomed the submission of proprietary or classified documents and trade secrets from any nation or corporation. Young described a few such documents and the unfavorable responses they had received. The British government objected to one document and attempted to have cryptome.org's Internet service provider shut the site down. Another document prompted diplomatic requests from the Japanese government for its removal. All attempts to shut the site down have thus far been rebuffed, but Young imagines that someone will eventually be successful.

Other information cryptome.org has received and published include proofs that American corporations used US intelligence to stay ahead of foreign competitors, the names of over 8,000 CIA informants, and, currently, the programs and keys associated with Russian programmer Dmitri Skylarov's crack of Adobe's E-book system, for which he was arrested in July.

An audience member asked what types of material cryptome.org would not publish. Young explained that cryptome.org is open to any kind of publication, but they have refused to publish child pornography documents and information related to biological warfare. They also feel that personal prerogative takes precedence over the public's right to know, so they will remove personal information and documents if requested by the person in question. They also reminded the audience that they do not verify the authenticity of the information they publish – they leave that to the interested reader.

Young repeatedly stressed what he believed to be the transitory nature of cryptome.org. He assured the audience that at some point cryptome.org will either be silenced or it will simply mature away from the cutting edge. When that finally happens, Young is confident that someone else will emerge at the vanguard of the quest to reverse the Panopticon state.

Designs against Traffic Analysis

Paul Syverson, U.S. Naval Research Laboratory

Summarized by Yong Guan

Paul Syverson used a pseudonym, "Peter Honeyman," on his talk, a joke which pervaded the rest of the conference.

Although the encryption of network packets ensures privacy of the payload in a public network, packet headers identify recipients, packet routes can be tracked, and volume and timing signatures are exposed. Since encryption does not hide routing information, public networks are vulnerable to traffic analysis.

Traffic analysis can reveal, for example, who is searching a public database, what Web sites are surfed, which agencies or companies are collaborating, where your email correspondents are, what supplies/quantities you are ordering and from whom, and so forth.

Knowing traffic properties can help an adversary decide where to spend resources for decryption and penetration. Therefore, it is important to develop countermeasures to prevent traffic analysis.

The security goal of traffic-analysis-resistant systems is to hide one or more of the following:

- Sender activity: that a site is sending anything
- Receiver activity: that a site is receiving anything
- Sender content: that a sender sent specific content
- Receiver content: that a receiver received specific content
- Source-destination linking: that a particular source is sending to a particular destination
- Channel linking: identifying the endpoints of a channel

Some systems were described:

Dining Cryptographers (DC) – networks, in which each participant shares secret coin flips with other pairs and announces the parity of the flips the participant has seen to all other participants and the receiver.

Chaum mixes – a network of mix nodes, in which messages are wrapped in multiple layers of public-key encryption by the sender, one for each node in a route. Most widely used anonymous communication systems use the Chaum mix method.


Paul Syverson

There are two kinds of routes for the messages: mix cascade, where all messages from any source move through a fixed-order "cascade" of mixes, and random route, where the route of any message is selected at random by the sender from the available mixes.

Remailers, mainly used for email anonymity, employ rerouting of an email through a sequence of multiple mail remailers before the email reaches the recipient, so that the true origin of the email can be hidden.

Anonymizer and SafeWeb provide fast, anonymous, interactive communication services. They are essentially Web proxies that filter out the identifying headers and source addresses from Web browsers' requests. Instead of the user's true identity (e.g., IP address), a Web server can only learn the identity of the Web proxy. Both offer encrypted links to their proxy (SSL or SSH). Anonymizer is a single point of failure, whereas SafeWeb is a double point of failure. SafeWeb offers additional protection from censorship.

Crowds aims at protecting users' Web-browsing anonymity. Like Onion Routing, the Crowds protocol uses a series of cooperating proxies (called jondo) to maintain anonymity within the group. Unlike Onion Routing, the sender does not determine the whole path. Instead, the path is chosen randomly on a hop-by-hop basis. At each hop a decision is made whether to submit the request directly to the end server or to forward it to another randomly chosen member according to forwarding probability. The expected path length is controlled by the forwarding probability. Cycles are allowed on the path. The receiver is known to any intermediate node on the route. Once a path out of a crowd is chosen, it is used for all the anonymous communication from the sender to the receiver within a 24-hour period. Crowds does not have a single point of failure and is a more lightweight crypto than mix-based systems. However, Crowds has limitations: all users must run Perl code, users have to have long-running high-speed Internet connections, an entirely new network graph is needed for a new or reconnecting Crowd member, connection anonymity is dependent on data anonymity, and responder protection is weak.

Onion Routing provides anonymous Internet connection services. The Onion Routing network operates on top of existing TCP/IP networks such as the Internet. It builds a rerouting path within a network of onion routers, which in turn are similar to real-time Chaum mixes. In Onion Routing, the data packet is broken into fixed-size cells, and each cell is encrypted multiple times (once for each onion router on the path). Thus, a recursively layered data structure called an onion is constructed. An onion is the packet transmitted along the rerouting path. The fixed size of an onion limits a route to a maximum of 11 nodes in the current implementation. Onions can be tunneled to produce arbitrary length routes.

Onion Routing I (Proof-of-concept) uses a network of five Onion Routing nodes operating at the Naval Research Laboratory. It forces a fixed length (five hops, i.e., five intermediate onion routers) for all routes.

Onion Routing II can support a network of up to 50 core onion routers. For each rerouting path through an Onion Routing network, each hop is chosen at random. The rerouting path may contain cycles, although only cycles with one or more intermediate nodes are allowed.

Freedom Network also aims at providing anonymity for Web browsing. From the user's point of view, Freedom is very similar to Onion Routing. Freedom consists of a set of nodes (called Anonymous Internet Proxy) which run on top of the existing Internet infrastructure. To communicate with a Web server, the user first selects a series of nodes to form a rerouting path and then uses this path to forward the requests to its destination. The Freedom Route Creation Protocol allows the sender to randomly choose the path, but the path length is fixed to be three. The Freedom client-user interface does not allow the user to specify a path-containing cycle. The Freedom client must either have all the intermediate nodes in the path chosen or choose a preferred first node and last node, and the intermediary nodes are picked at random.

For more information, visit *http://www.onion-router.net* and *http://www.syverson.org*.

Question: Who manages the onion routers? Are they managed independently?

Answer: Yes. The onion routers can be distributed anywhere and be managed by different groups.

Question: Do you believe that, the longer the path, the safer the anonymous communication system?

Answer: I am not sure.

Countering SYN Flood Denial-of-Service (DoS) Attacks

Ross Oliver, Tech Mavens

Summarized by David Richard Larochelle

SYN flood attacks are a nasty DoS attack. The attacker sends a SYN packet but does not complete the three-way handshake. This is hard to defend against because SYN packets are part of normal traffic, and unlike ping attacks you can't firewall them. Since SYN packets are small, the attack can be done with limited bandwidth. Finally, the attacks are difficult to trace because source IP addresses can be faked. Ross Oliver stressed that it's up to you to defend yourself (law enforcement is unable to deal with attacks as they occur, if they can deal with them at all) and suggested that firewalls employing SYN flood defenses are the best way of doing this.

He reviewed four such products: PIX by Cisco, Firewall-1 by Checkpoint, Netscreen 100 by Netscreen, and AppSafe (previously called AppSwitch) by TopLayer. To test these products, he placed a Web server behind the firewall and used a machine with a script which called wget repeatedly to request Web pages to represent the legitimate client traffic. An attacking machine threw SYN packets with forged source addresses at the Web server.

The Cisco PIX used a threshold technique which allowed a set number of incomplete connections and dropped additional SYN packets. The tests showed no significant improvement over no firewall. The Firewall-1 fared slightly better. It lets SYN packets reach the Web server and then sends an ACK packet to the Web server to complete the three-way handshake. Under a SYN flood attack, the Web server will then have a bunch of completed connections instead of half-open ones. Firewall-1 protected up to 500 SYNs per second but with degraded response time. The Web server returned to normal 3–10 minutes after the attack ceased.


Ross Oliver

Netscreen and AppSafe had the best results. If these firewalls detect a SYN flood attack, they proxy the incoming connections and only send the Web server the SYN and ACK packets if the handshake is completed by the client. Netscreen detects SYN floods by looking at the number of incomplete connections. It protected up to 14,000 SYNs/sec with acceptable response times and continued to function at higher SYN rates but with increasing delays. The server responded normally immediately after the attack.

AppSafe used a more elaborate approach. It determined whether to proxy a connection request based on the source IP address. SYN packets from IP addresses which had recently behaved legitimately were let through to the Web server immediately. Only connections from previously unseen or malicious IP addresses were proxied. AppSafe was effective up to 22,000 SYNs/sec, which was the most traffic that the attacking machine could produce in this test. However, it was pointed out that, in the test, the client machine used only one IP. This technique may not work as well in a situation in which there are new connections from previously unseen clients.

How much protection you need depends on what type of attack you expect. An attacker with a Cable or DSL connection can produce 200 SYNs/sec. An attacker with a T1 can produce 2,343 SYNs/sec. According to the paper "Inferring Internet Denial-of-Service Activity" presented the previous day, 46% of DoS attacks involved more than 500 SYNs/sec but only 2.4% were above 14,000 SYNs/sec. This level can be handled with a single firewall. Multiple or distributed attacks may require multiple parallel firewalls. Because of the wide range of performance between devices, Oliver stressed the importance of testing and advised testing the devices yourself if possible.

Real Stateful TCP Packet Filtering with IP Filter

Guido van Rooij, Eindhoven University of Technology

Summarized by Evan Sarmiento

Old firewall implementations used to filter TCP sessions using addresses and ports only, creating an interesting problem. The administrator would have to guess the source port of the packet in order to filter it correctly. In order to solve this, a new trend in firewalls is to introduce stateful packet filtering. Stateful packet filters remember and only allow through addresses and ports of connections that are currently set up.

Even before Guido van Rooij's work, IP Filter did have stateful packet filtering, but it was implemented in the wrong way. IP Filter does take sequence, ACK, and window values into account, but it makes the wrong assumption that packets seen by the filter host will also be seen by the final destination. This assumption caused IP Filter to drop packets in certain situations. The new state engine for IP Filter encompasses the following goals:

- Conclusions made by the engine must be provable.
- All kinds of TCP behavior must be taken into account.
- The number of blocked packets must be minimized.
- Blocking of packets must never lead to hanging connections.
- Opportunities for abuse should be made as small as possible.

The new state engine includes 20 bytes per state entry and about 40 lines of C code without loops; thus, the performance overhead is minimal.

However, even the new state engine is not always successful, even though it is a great improvement. Occasionally, blocked FIN and ACK packets cause problems in the state timeout handling for TCP half-closed sessions. IP Filter drops packets coming from a few Windows NT workstations for a strange and as yet unknown reason.

Guido then outlined some future additions to IP Filter. He would like to be able to fix fragment handling, add support for sessions entering the state table after establishment, and check validity of a session if a packet comes in from the middle of the connection.

REFEREED PAPERS

SESSION: DENIAL OF SERVICE

Summarized by Stefan Kelm

Using Client Puzzles to Protect TLS

Drew Dean, Xerox PARC; Adam Stubblefield, Rice University

Adam Stubblefield presented their work on a DoS protection technique, namely, the use of client puzzles within the TLS protocol. Even though client puzzles have been supposed to be a solution to DoS attacks, Stubblefield pointed out the lack of actual implementations. The choice of TLS as the protocol to protect against DoS seems obvious, but TLS is subject to DoS attacks because of the computing-expensive cryptographic operations performed at both the client and the server side. A server, however, has to perform the more expensive RSA decrypt operations during the session handshake; thus any small number of clients could easily overload a TLS server by flooding the server with TLS handshake messages. The goal of this work is to prevent this using cheap methods.

The idea of TLS-based cryptographic puzzles is to first let the client do the work, and subsequently the server. If the server is under a heavy load it sends a so-called "puzzle request" to the client. The client, in turn, has to compute a number of operations which it then uses to send a "puzzle solution" back to the server. Thus, the server will not need to continue the TLS handshake unless the client has proven its intent to really open a TLS connection.

Client puzzles are surprisingly easy to implement on both the client and the server side. Stubblefield used modified OpenSSL and mod_ssl source code to test the implementation. The implementation uses a metric which tracks unfinished RSA decrypt requests in order to decide whether or not the server is assumed to be under attack. Since adding more latency to the TLS protocol was not a goal of this work, the server only sends a puzzle request back to the client if it really has to. This is implemented by using variable thresholds.

The author concluded that they are able to protect against certain denial-of-service attacks at not much cost and with a good user experience. Moreover, the proposed solution can be implemented using already existing code.

For more information, contact *astubble@rice.edu*.

Inferring Internet Denial-of-Service Activity

David Moore, CAIDA; Geoffrey M. Voelker and Stefan Savage, University of California, San Diego

This paper, awarded the best paper award, tried to answer the question of how prevalent denial-of-service attacks in the Internet currently are. The authors ran a test over a period of three weeks, trying to come up with an estimate of worldwide DoS activity.

David Moore presented the so-called "backscatter analysis" as their key idea and outlined the basic technique: since attackers normally use spoofed source IP addresses, the "real owners" of those IP addresses regularly receive response packets from the systems being attacked (Moore called these "unsolicited responses"). By monitoring these unsolicited responses one is able to detect different kinds of DoS attacks. Furthermore, by observing a huge number of different IP addresses over a longer period of time, sampling the results can provide an overview of attacks going on.

Moore presented some interesting results and displayed a number of figures and tables showing the number of attacks, the attacks over time, the attack characterization, the attack duration distribution, and the attack rate distribution. Moore's team observed a number of minor DoS attacks (described as "personal vendettas") as well as some victims under repeated attack. Classifying the victims by TLD showed countries like Romania and Brazil being attacked far more often than most other TLDs. The presenter's hypothesis was that either those countries host ISPs that attack each other, or there simply are more hackers located in Romania and Brazil (this was later denied by someone in the audience stating that Romania has really nice people).

In conclusion, the authors observed some very large DoS attacks, though most attacks seem to be short in duration. Another result showed the majority of attacks being TCP based. To clarify, this technique is not good at distinguishing between DoS and DDoS

attacks since it is not good at distinguishing between attackers.

During the Q&A session, one question was on why the analysis showed no attacks on the .mil domain. The response given was that either .mil is not under attack (unlikely) or that backscatter packets are being filtered.

For more information, contact *dmoore@caida.org*, or see *http://www.caida.org/outreach/papers/backscatter/*.

MULTOPS: A Data-Structure for Bandwidth Attack Detection

Thomer M. Gil, Vrije Universiteit/MIT; Massimiliano Poletto, MIT

Thomer Gil proposed a heuristic as well as a new data structure to be used by routers and similar network devices to detect (and possibly eliminate) denial-of-service attacks. Most DoS attacks show disproportional packet rates with a huge number of packets being sent to the victim and only very few packets being sent by the victim in response. The new data structure, called MULTOPS (Multi-Level Tree for Online Packet Statistics), monitors certain Internet traffic characteristics and is able to drop packets based on either the source or the destination address.

The main implementation challenges with MULTOPS have been and still are the precise identification of malicious addresses, athe achievement of a small memory footprint, and a low overhead on forwarding "real" traffic as opposed to DoS-based traffic. MULTOPS is implemented as a memory-efficient tree of nodes which contains packet-rate statistics and which dynamically grows and shrinks with the traffic being observed. At the current implementation, packets are dropped based on either a variable packet rate or a ratio. Since it usually is impossible to identify an attacker (because of IP spoofing), packets can be dropped based on the victim's IP, too.

The authors succeeded in simplifying memory management and the mechanism that keeps track of packets. Gil pointed out that their solution is successfully being used by a network company. They are currently trying to focus on the behavior of different TCP implementations as well as protocols other than TCP.

Someone brought up the question of differentiating DoS traffic from traffic that normally shows disproportional packet flows, e.g., video traffic. The reply suggested the possibility of building some kind of knowledge base. A lively discussion on random class A addresses within MULTOPS subsequently arose but was taken offline.

For more information, contact *thomer @lcs.mit.edu*.

SESSION: HARDWARE

Summarized by Anca Ivan

Data Remanence in Semiconductor Devices

Peter Gutmann, IBM T.J. Watson Research Center

Peter Gutmann explained the dangers of deleting data in semiconductors. Everyone knows that deleting data from magnetic media is very hard, but not too many realize that the same problem exists for semiconductors, especially since there are so many ways of building semiconductors, each with its own set of problems and solutions. After giving a short background introduction in semiconductors and circuits (n-type, p-type, SRAM, DRAM), Peter described some of the most important issues:

- Electromigration: because of high current densities, metal atoms are moved in the opposite direction of the normal current flow. The consequence is that the operating properties of the device are strongly altered.
- Hot carriers: during operation, the device heats up and its characteristics change considerably.
- Ionic contamination: this is no longer an issue and its effects are no longer significant.
- Radiation-induced charging: it freezes the circuit into a certain state.

The first phenomenon enables attackers to recover partial information from special-purpose devices (e.g., cryptographic smartcards). The next two can be used to recover data deleted from memory. In order to avoid long- and short-term data retention from semiconductors (a DES key was recovered in the '80s), researchers developed a series of solutions that use various semiconductor forensic techniques, including the following two:

- Short-term retention: probably the safest way to defend against it is not to keep the same values in the same memory cells for too long (maximum a few minutes).
- Long-term retention: in 1996, some researchers proposed periodically flipping the stored bits. In this way, no cell holds the same bit value for long enough to "remember" it.

In the end, Peter talked about how all the problems cited above extend to flash memory. For example, random generators can generate strings of 1s when the pool is empty, or information can be leaked into adjacent cells into shared circuitry.

Even though the entire presentation scared at least one person in the audience (guess who?), Peter assured us that reality is not that gloomy. In fact, the only problem is the lack of a standard. Every time people decide to choose one implementation method, they should also choose which solutions are best for it. Answering a question, Peter told us that personal computers are not affected

by those problems but that most specialized devices, like airplane black boxes, can leak information if analyzed with very sophisticated equipment. But then again who has such equipment?

StackGhost: Hardware-Facilitated Stack Protection

Mike Frantzen, CERIAS; Mike Shuey, Purdue University

The authors presented a software solution to the return-pointer hijacking problem. The most important step in the function-call process is when the caller saves the return pointer before giving the control to the called function. Many attacks are based on changing this pointer. When the callee finishes, the return pointer dictates which function takes control next. StackGhost is a piece of software that automatically and transparently saves the return pointer and replaces it with another number. When the called function completes, StackGhost verifies the integrity of that number (catching, in this way, possible attacks) and reinstalls the correct pointer value.

The security of StackGhost depends on how it modifies the return pointer to catch attacks; the authors have tried several ways:

- Per kernel XOR with a 13-bit signed cookie: the main problem is that an attacker can find out the cookie by starting several arbitrary programs.
- Per process XOR with a 32-bit cookie: this is safer than the previous method, but more expensive.
- Encrypt/decrypt the return pointer: this method seems to be the most expensive.
- Return-address stack: this method replaces the return pointer with another number and saves the pointer into a return-address stack. However, this would impede other applications from running correctly.

After making performance measurements for all techniques, the authors noticed that the chances of StackGhost not catching an attack were 1 in 3 for XOR cookies and 1 in 2^{32} for the return-address stack. The conclusion was that StackGhost was offering protection against return-pointer overriding to all processes in the system (which might be seen as a disadvantage).

Improving DES Coprocessor Throughput for Short Operations

Mark Lindemann, IBM T.J. Watson Research Center; Sean W. Smith, Dartmouth College

While the first two talks in this session were at opposite poles (one deeply hardware and one purely software), the third one was somehow in the middle. The presenter, Sean Smith, is one the fathers of the cryptographic card developed at IBM and presently working at Dartmouth College. Everything started in a very optimistic fashion, with the usual introduction we would have expected from an IBM representative trying to sell us this device: "It is secure . . . it is fast . . . it is reliable." All the buzzwords were there. However, with the next slide this changed to "It is not as secure . . . fast . . . as we thought." For example, the specification promised the DES speed to be 20 megabytes/second when in reality a friend obtained less than two kilobytes/second in a database application. Where was the discrepancy coming from? The main intuition was that the specification gives the performance for operations on megabytes of input. The real speed is much slower if the data is shipped to the card in small chunks. The difference between specs and reality was too big not to be studied, and Lindemann decided to find out the reasons behind it. First, they built a model that simulates the database application and then tried to improve the speed by modifying the execution conditions in the following ways:

- Reducing card-host interaction: "folklore in IBM" taught them that any card-host interaction consumes too much time. Thus, they rewrote the application to minimize the number of interactions. The speed went up to 18–23 kilobytes/second; however, it was still too far from megabyte speed.
- Batching all operations into one chip operation: chip resets were too expensive. The speed became 360 kilobytes/second.
- Batching into multiple chip operations: it reduced the number of Layer 3 – Layer 2 switches. The speed changed to 30–290 kilobytes/second, still not good.
- Reducing data transfers: they did it by using an internal key-table and boosted the speed to 1,400 kilobytes/second.
- Using memory-mapped I/O: this eliminated the internal ISA bus bottleneck. The speed went up to 2,500 kilobytes/second.
- Batching operation parameters: instead of sending them as separate packets. It increased the speed to 5000 kilobytes/second. This was even more than they were expecting, but the results were incorrect. The client had asked for speed but hadn't mentioned anything about correctness. So was the problem solved?
- Not using memory-mapped I/O: to increase accuracy, they gave up on memory-mapped I/O for initialization vectors and count. Unfortunately, there was a small performance cost: the speed was now 3,000 kilobytes/second.

From the client's point of view, all of these steps showed them that the only way to maximize the performance while using the secure coprocessor was to design DES-batched API. From the designer's point of view, the conclusions

were simpler: always distrust folklore and think if and how people will use your product before designing it!

SESSION: FIREWALLS/INTRUSION DETECTION

Summarized by Stefan Kelm and Yong Guan

Architecting the Lumeta Firewall Analyzer

Avishai Wool, Lumeta

"What is your firewall doing?" Avishai Wool asked the audience at the beginning of his presentation, thereby describing the motivation to build LFA, the "Lumeta Firewall Analyzer."

Firewalls have been installed by almost all companies connected to the Internet. However, the underlying policy often is far from being good enough to actually protect the company from outside attackers. Network administrators often do not know how to set up a firewall securely, much less how to test or audit the firewall configuration. Wool pointed out that LFA is the successor of the Fang prototype system built at Bell Labs as a firewall analysis engine.

The key idea is not to probe the actual firewall in any way but to allow testing of the configuration before the firewall is deployed. The firewall's routing table and configuration files are used as input to the LFA, which parses these files and simulates the behavior of any possible packet flow combination (LFA mainly offers support for Firewall-1 and PIX). The results are presented to the user as HTML pages.

Wool concluded by giving a short demonstration. As input to the LFA, he used a short Firewall-1 policy which contained only six rules and explained why even such a short rule set might lead to problems once the firewall is deployed. During the Q&A session he emphasized that the LFA only checks packet headers, not the content, and cannot therefore detect tunneling problems.

For more information, contact *yash@acm.org*, or visit *http://www.lumeta.com/firewall.html*.

Transient Addressing for Related Processes: Improved Firewalling by Using IPv6 and Multiple Addresses per Host

Peter M. Gleitz and Steven M. Bellovin, AT&T Labs-Research

The authors proposed a method to simplify firewall decisions. By using the large address space brought by IPv6, they employed a strategy of multiple network addresses per host. That is, for each request on the client host an IPv6 address is tied to the client process. The firewall now makes access decisions based on transport layer protocol information (i.e., filtering is shifted from ports to addresses). Once approved, the firewall allows all traffic between the two peers to pass to and fro. Once the service is finished the IPv6 address is discarded. This method is called TARP (transient addressing for related processes). TARP employs two different types of addresses: (fixed) server addresses and process group addresses.

Gleitz discussed how TARP works with TCP and UDP applications and with the firewall, router, domain name server, and IPSEC. Employing TARP does not necessarily affect the routers, though TARP-aware routers can perform better. Moreover, Gleitz pointed out that no modifications to standard applications such as Telnet, SSH, FTP, Sendmail, or TFTP are necessary in order to use TARP. He also mentioned briefly some interop problems with protocols such as DNS and ICMPv6.

For more information, contact *pmgleit@netscape.net.*

Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics

Mark Handley and Vern Paxson, ACIRI; Christian Kreibich, Technische Universität München

This paper focused on the problem of network intrusion detection system (NIDS) evasion. Attackers usually can fool any NIDS by exploiting certain ambiguities in the packet flow being monitored by the NIDS, i.e., (1) the NIDS may lack complete analysis of the packet flow (e.g., no TCP stream reassembly); (2) the NIDS may lack end-system knowledge (e.g., certain application vulnerabilities); and (3) the NIDS may lack network knowledge (e.g., the topology between the NIDS and an end system).

As a solution, Paxson proposed the deployment of a "normalizer," the goal of which would be to observe all packets being sent between two network nodes (he called that a "bump-in-the-wire") and to modify ("normalize") packets that seem to be ambiguous for one reason or another. As an example the author described problems with two overlapping fragments: the normalizer would re-assemble (and re-fragment, if necessary) those packets before forwarding. Since re-assembly is a valid operation, the normalizer would, in this example, have no impact on the semantics at all.

Paxson also pointed out some of the problems with this approach, one of which is the "cold start" problem: (re-)starting the normalizer will show many valid connections already established. It is difficult to handle those connections accordingly (this is also true for the NIDS itself). The normalizer has been implemented and will be available at *www.sourceforge.net* soon.

In the Q&A session Steven Bellovin wanted to know whether normalization

would not be needed at the application layer as well. The presenter answered in the affirmative.

For more information, contact *vern@aciri.org*.

SESSION: OPERATING SYSTEMS

Summarized by Mike Vernal

Security Analysis of the Palm Operating System and Its Weaknesses against Malicious Code Threats

Kingpin and Mudge, @stake, Inc.

Kingpin and Mudge began their presentation with a bold fashion statement, appearing in matching white bathrobes. Their bathrobes aimed to underscore the fact that PDAs can undermine user privacy in a public setting. Their efforts were later rewarded with the coveted USENIX Style Award, presented by the real Peter Honeyman, of the University of Michigan.

Kingpin and Mudge

The presentation centered on the security threat posed by the recent ubiquity of Personal Digital Assistants (PDAs), and, more specifically, devices running the Palm Operating System. Palm devices increasingly are being used in security-sensitive settings such as hospitals and government agencies. While the government is now aware of the security threat posed by PDAs, the corporate world has remained generally oblivious.

The security threat of the Palm stems from the PalmOS's lack of a well-defined security framework. Specific weaknesses enumerated include the direct addressability of hardware, the lack of memory encryption, the lack of ACLs, weak obfuscation of passwords, and a backdoor debug mode that allows for the bypassing of "system lockout." Because of these and other weaknesses, the audience agreed with the assertion that developing a secure application on top of the PalmOS would be impossible.

The presentation suggested that unscrupulous users could exploit a number of weaknesses to install malicious code, including normal application installation, desktop conduits, creator ID replacement, wireless communications, and the Palm Debugger. Another threat raised by Kingpin and Mudge was the possibility of a new set of cross-pollinating viruses, which could be acquired via a Palm and propagate themselves to desktop computers via the HotSync operation, or vice versa.

With the growing popularity of PDAs, Kingpin and Mudge invoked Occam's Razor: all other factors being equal, the PDA may be the malicious user's easiest point of entry into an information network. The upcoming PalmOS 4.0 reportedly fixes some of the security concerns raised. In the interim, users should be made aware of the possible security threats and restrict or eliminate their use of sensitive data and applications on Palm devices.

Secure Data Deletion for Linux File Systems

Steven Bauer and Nissanka B. Priyantha, MIT

Steven Bauer presented an implementation of a kernel-level secure data-deletion (SDD) mechanism for the ext2 file system.

Bauer suggested that with the increasing prevalence of public kiosks, thin clients, multi-user computing clusters, and distributed file systems, users will want to ensure that when their data is deleted from these systems, it is truly and irretrievably deleted.

In 1996, Peter Gutmann of IBM demonstrated that data that had been overwritten on a magnetic disk could be recovered using advanced probing techniques. While popular lore has suggested certain government agencies may be able to recover data overwritten dozens of times, no commercial data recovery company contacted in conjunction with Bauer's research believed that it could recover data that had been overwritten more than once. As such, the SDD system as described probably only needs to overwrite data a few times.

This SDD system was designed to ensure that all flagged data is deleted, even in the event of system failure. The deletion process was designed as an asynchronous daemon to ensure that it did not interfere with normal operation and performance. Though implemented for the ext2 file system, Bauer asserts that this system should be portable to any block-oriented file system.

The ext2 implementation used the unused secure-deletion flag, settable with the chattr() function. With this mechanism, the granularity with which secure deletion can be specified ranges from an entire device to an individual file. Questions were raised as to the vulnerability of temporary files that are not flagged in a secure deletion zone. Bauer recommended that for maximum security, the entire device should be flagged for secure deletion.

SESSION: MANAGING CODE

Summarized by Sameh Elnikety

Statically Detecting Likely Buffer Overflow Vulnerabilities

David Larochelle and David Evans, University of Virginia

Buffer overflow attacks account for approximately half of all security vulnerabilities. Programs written in C are particularly susceptible to buffer overflow attacks because C allows direct pointer manipulations without any bounds checking.

Run-time approaches to mitigate the risks of buffer overflow incur performance penalties, and they turn buffer overflow attacks into denial-of-service attacks by terminating execution of the attacked processes. Static checking overcomes these problems by detecting likely vulnerabilities before deployment.

The authors developed a practical lightweight static analysis tool based on LCLint to detect a high percentage of likely buffer overflow vulnerabilities.

The tool exploits semantic comments (annotations) that describe programmer assumptions and intents. These annotations are treated as regular C comments by the compiler but are recognized as syntactic entities by LCLint. The annotations represent preconditions and postconditions for functions to determine how much memory has been allocated for buffers. LCLint uses traditional compiler data flow analyses with constraint generation and resolution. Also, LCLint uses loop heuristics to efficiently analyze many loop idioms in typical C programs.

The authors used the tool to analyze wu-ftpd, which is a popular open source FTP server, and part of BIND, which is a set of domain-name tools and libraries that is considered the reference implementation of DNS. Running LCLint is similar to running a compiler. For wu-ftpd, it took less than one minute for LCLint to analyze all 17,000 lines of unmodified wu-ftpd source code. This resulted in 243 warnings that showed known and unknown buffer overflow vulnerabilities.

LCLint source code and binaries are available from *http://lclint.cs.virginia.edu*.

FormatGuard: Automatic Protection from printf Format String Vulnerabilities

Crispin Cowan, Matt Barringer, Steve Beattie, Greg Kroah-Hartman, WireX Communications, Inc.; Mike Frantzen, Purdue University; and Jamie Lokier, CERN

In June 2000, a major new class of vulnerabilities called format bugs was discovered when a vulnerability in WU-FTP appeared that looked almost like a buffer overflow but was not. It is unsafe to allow potentially hostile input to be passed directly as the format string for calls to printf-like functions. The danger is that the inclusion of % directives, especially %n, in the format string coupled with the lack of any effective type or argument counting in C's varargs facility allows the attacker to induce unexpected behavior in programs.

The authors developed FormatGuard, a small patch to glibc. It provides general protection against format bugs using particular properties of the GNU CPP macro-handling mechanism to extract the count of actual arguments to printf statements. This is then passed to a safe printf wrapper. The wrapper parses the format string to determine how many arguments to expect, and if the format string calls for more arguments than the actual number of arguments, it raises an intrusion alert and kills the process.

FormatGuard fails to protect against format bugs under several circumstances. For example, if the program uses a function pointer that has the address of printf, then it evades the macro expansion.

FormatGuard is incorporated in WireX's Immunix Linux distribution and server products. It is available as a GPL'd patch to glibc at *http://immunix.org*.

Detecting Format String Vulnerabilities with Type Qualifiers

Umesh Shankar, Kunal Talwar, Jeffrey S. Foster, and David Wagner, University of California, Berkeley

Systems written in C are difficult to secure, given C's tendency to sacrifice safety for efficiency. Format string vulnerabilities can occur when user input is used as a format specifier. One of the most common cases is when the program uses printf with one argument: a user-supplied string assuming that the string does not contain any % directive. The authors presented a tool (cqual) that automatically detects format string bugs at compile time using type-theoretic analysis techniques. With this static analysis, vulnerabilities can be proactively identified and fixed before the code is deployed.

Cqual builds an annotated Abstract Syntax Tree (AST). Then, it traverses the AST to generate a system of type constraints, which is solved online. Warnings are produced whenever an inconsistent constraint is generated. Cqual presents the results of tainting analysis to the programmer using Program Analysis Mode for Emacs (PAM). PAM is a GUI that is designed to add hyperlinks and color mark-ups to the preprocessed text of the program. The interface shows the taint flow path to help programmers determine how a variable becomes tainted.

The configuration files makes cqual usable without modifying the source code. The authors analyzed four security-sensitive benchmark programs with the same standard prelude file and no

direct changes to the applications' source code. Typically a few application-specific entries were added to the prelude file to improve accuracy in the presence of wrappers around library functions. Cqual reliably finds all known bugs for the benchmark programs. It also reports few false positives. Cqual is fast; it usually takes less than a minute.

Cqual is available at *http://bane.cs.berkeley.edu/cqual*.

SESSION: AUTHORIZATION

Summarized by Rachel Greenstadt

Capability File Names: Separating Authorization from User Management in an Internet File System

Jude T. Regan, consultant; Christian D. Jensen, Trinity College

On the Internet there is no reliable way to establish an identity. Flexible user-user collaboration outside of an administered system so that people could create ad hoc work groups and remove arbitrary limitations to information sharing is the authors' goal.

Such a system should be globally accessible, easy to use, and require as little intervention by system administrators as possible. This system should integrate with existing systems and applications. It should have fine granularity so that users would not have to use complicated export mechanisms to share files.

The authors used the concept of a capability, a token conveying specified access rights to a named object in order to make the identity of the object and the access rights inseparable. They embedded the capability in something every system knows – the file name.

The authors concluded that the system was safe from interception and modification Attackers could forge the client part but not the server part of the file names. Service could be interrupted, but protecting against this is impossible without complete control of the network. Performance was evaluated in comparison to NFS. Most of the overhead was in the open statement. Reads were slightly slower and writes were much slower, but they felt this could be alleviated by implementing symmetric writes.

Kerberized Credential Translation: A Solution to Web Access Control

Olga Kornievskaia, Peter Honeyman, Bill Doster, and Kevin Coffman, CITI, University of Michigan

There are two different authentication mechanisms: those used for services such as login, AFS, and mail, for which Kerberos is popular, and public key-based mechanisms such as SSL, which is used to establish secure connections on the Web. These systems need to be able to work together to satisfy a request.

The authors propose to achieve the best of both worlds by leveraging Kerberos to solve PKI key management. This will use existing infrastructures which allow strong authentication on the Web with SSL and which provide access to Kerberized back-end services. They propose a system to provide interoperability between PKI and Kerberos. Their system consists of (1) a Certificate Authority (CA), KX509, which creates short-lived certificates, (2) a Web server which acts like a proxy for users by requesting services from Kerberized back-end services and (3) a Kerberized Credential Translator, which translates public-key credentials to Kerberos. They created a prototype of their system called WebAFS using AFS as an example Kerberized service.

Dos and Don'ts of Client Authentication on the Web

Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster, MIT

[This paper received the Best Student Paper Award]

Kevin gave a very amusing presentation which illustrated the gap between security theory and practice. He described a variety of Web sites that used insecure client authentication schemes and presented hints on how to avoid their mistakes.

Client authentication seems like a solved problem, but many sites continue to come up with homebrew schemes which just don't quite get it right. Out of the 27 Web sites the cookie eaters group examined, they weakened the security on two sites, were able to mint authenticators on eight, and on one site were able to obtain the secret key. Some of these sites were high profile, such as the *Wall Street Journal* (wsj.com), Sprint PCS (sprintpcs.com), and FatBrain (fatbrain.com).

In most cases, the mistakes made in these sites were simple. By simply looking at their cookie files the authors could query Web servers and look at headers, responses, and create sample authenticators. Except for Sprint, these attacks involved no eavesdropping at all. The schemes were not even strong against what the authors termed the "interrogative adversary." This adversary has no special access, but it adaptively queries a Web server a reasonable number of times. It just sits there and connects to port 80; it cannot defeat SSL client authentication, HTTP basic, or digest authentication. The best such an adversary can do against a pass-

Kevin Fu

word sent in the clear is a dictionary attack. However, some homebrew cookie schemes are vulnerable.

In the case of the *Wall Street Journal*, a site with half a million paid subscribers who can track their stocks and buy articles, the authors found that the makers of the site had misused cryptography and created an authenticator weaker than a plaintext password.

Some hints provided for client authentication were: limit the lifetime of authenticators since browsers cannot be trusted to expire cookies; expiration dates must be cryptographically signed (this was another problem with *WSJ*). Authenticators should be unforgeable, and cookies should not be modifiable by the user. There should be no bypassing of password authentication. Digital signatures are great, but you should not allow the things you sign to be ambiguous. For example, the concatenation of "Alice, 21-Apr" and "Alice2, 1-Apr" is the same. Delimiters can help solve this problem. He presented a simple scheme for building an authenticator which would work against the interrogative adversary.

In summary, there are many broken schemes out there, even in popular Web sites. There are even more juicy details in the authors' technical report. Cookie schemes are limited; live with it or move on. You can join the authors by donating your cookies for analysis at *http://cookies.lcs.mit.edu*.

SESSION: KEY MANAGEMENT

Summarized by Sameh Elnikety

SC-CFS: Smartcard Secured Cryptographic File System

Naomaru Itoi, CITI, University of Michigan

Storing information securely is one of the most important applications of computer systems. Secure storage protects the secrecy, authenticity, and integrity of the information. SC-CFS implements a secure file system and is based on Matt Blaze's Cryptographic File System for UNIX (CFS). SC-CFS uses a smartcard to generate a key for each file rather than for each directory. The per-file key encryption counters the password-guessing attack and minimizes both the damage caused by physical attack, compromised media, and bug exploitation.

When an encrypted file is updated, a new key is generated for that file and the file is re-encrypted for increased security. SC-CFS employs the same authentication mechanism as CFS, using an encrypted signature containing both a random number and a predefined sequence. A signature is stored in each directory. When a user starts to access a directory, SC-CFS gets the user key and decrypts the signature to recover the predefined sequence. If the sequence is not recovered, SC-CFS denies the user access to the directory.

SC-CFS is more secure than CFS because the master key is a random number instead of a password. This prevents dictionary attacks. Also, the user master key is not exposed to the host, and a stolen file key would reveal only one file and then only until that file is updated and consequently re-encrypted with a new file key.

The author implemented SC-CFS as an extension to CFS, then evaluated the performance of SC-CFS in comparison with CFS and a local Linux file system (ext2) using the Andrew Benchmark test. The results show that the performance of the system is not yet satisfactory because smartcard access is the bottleneck of SC-CFS. SC-CFS works as efficiently as ext2 and CFS when it does not access a smartcard. However, SC-CFS is significantly slower than CFS when it accesses a smartcard because the smartcard generates a key in 0.31 seconds.

Secure Distribution of Events in Content-Based Publish Subscribe Systems

Lukasz Opyrchal and Atul Prakash, University of Michigan

Some Internet applications, such as wireless delivery services and inter-enterprise supply-chain management applications, require high scalability as well as strict security guarantees. The content-based publish subscribe paradigm is one of the messaging technologies that facilitate building more scalable and flexible distributed systems. In the publish subscribe model, publishers publish messages and send them to subscribers via brokers. Each broker manages a large number of subscribers. The broker encrypts every message and broadcasts it to subscribers. The broker needs to guarantee the confidentiality of the messages so that only a specific group of subscribers can read the message.

Each subscriber has an individual symmetric pair key shared only with its broker. A naïve way to achieve this secure end-point delivery is for the broker to encrypt each message with a new key. Then, the broker sends the new key securely to each subscriber in the target group, by encrypting the new key with the symmetric key shared between the broker and the subscriber. The number of encryptions limits the broker throughput and system scalability. For the naïve approach, the number of encryptions is the same as the group size.

The authors presented four caching strategies to reduce the number of required encryptions. Simple cache assumes that many messages will go to the same subset of subscribers. Simple cache creates a separate key for each group and caches it. Build-up cache is based on the observation that many groups are subsets of other larger groups. Build-up cache uses a heuristic

to select some groups to cover the target group. Clustered cache uses a much smaller cache size by dividing the subscribers into clusters. Then, it uses the simple-cache method to send a message to the target subgroup in each cluster. Clustered-popular cache maintains both a simple cache and a clustered cache. When a new message arrives, clustered-popular cache searches for the target group in the simple cache. If the group is not found it uses the clustered cache to send the message to the appropriate subgroup in each cluster.

The authors analyzed the four caching strategies to find the average number of required encryptions and ran a number of simulations to confirm the theoretical results. They found that clustering the subscribers can substantially reduce the number of encryptions, which can be further reduced by adding a simple cache to clustered cache. Build-up cache, however, has little effect on the number of required encryptions.

A Method for Fast Revocation of Public Key Certificates and Security Capabilities

Dan Boneh, Stanford University; Xuhua Ding and Gene Tsudik, University of California, Irvine; Chi Ming Wong, Stanford University

The authors presented a new approach to fast certificate revocation using an online semi-trusted mediator (SEM). Suppose an organization has a Public Key Infrastructure that allows users to encrypt and decrypt messages and to digitally sign the messages. If an adversary compromises the private key of a user, then the organization needs to immediately prevent the adversary from signing or decrypting any message.

The overall architecture of the system is made up of three components. First, the central Certificate Authority (CA) generates a public key and a private key for each user. The private key consists of two parts. The CA gives the first part only to the user, and the other part only to the SEM. Second, the SEM responds to user requests with short tokens. The tokens reveal no information to other users. Third, the user contacts the SEM in case he wants to generate a digital signature or to decrypt a message. The system uses the MRSA encryption technique, which is similar to RSA, in a way that is transparent to peer users. The encryption process is identical to standard RSA. For the decryption process, the SEM does part of the decryption and the user does the remaining part. Both the SEM and the user must perform their share to decrypt a message. Digital signatures are generated in a similar way to performing decryption.

The authors implemented the system using OpenSSL and provided a client API and server daemons. The performance measurements showed that signature and encryption times are essentially unchanged from the user's perspective. The authors also implemented a plug-in for Eudora that enables users to sign their emails using the SEM. This approach achieves immediate revocation of public key certificates and security capabilities for medium-size organizations rather than the global Internet.


Gene Tsudik

The implementation of the system is available at:
http://sconce.ics.uci.edu/sucses.

The SEM Eudora plug-in is available at:
http://crypto.stanford.edu/semmail.

SESSION: MATH ATTACKS!

Summarized by Kevin Fu

PDM: A New Strong Password-Based Protocol

Charlie Kaufman, Iris Associates; Radia Perlman, Sun Microsystems Laboratories

A bright and cheery Radia Perlman talked about Password-Derived Moduli (PDM), a protocol useful for both mutual authentication and securely downloading credentials. PDM's notable features and improvements over existing protocols include unencumberance by patents, better overall server performance, and better performance when not storing password-equivalent data on the server.

Despite the promise of smartcards, passwords are still important for authentication. Demonstrating this importance, Perlman cited her own habit of misplacing any hardware token given to her. However, she can remember a password.

PDM deterministically generates a prime from a user's password and salt such as the username. To generate a prime, the user Alice fills out chunks of the right size with the hash of ("Alice," password, constant). PDM then searches for a safe "Sophie Germain" prime (p). A prime is Sophie Germain if (p-1)/2 is also a prime. PDM then uses this prime as the modulus in Diffie-Hellman exchanges.

PDM is potentially fast on a server and tolerably slow on a client. Although 512-bit Diffie-Hellman moduli are within the realm of breakability, a dictionary attack against PDM requires a Diffie-Hellman exponentiation per password guess. This places a lot of computational burden on an adversary. Using 512-bit moduli instead of 1024-bit moduli improves performance on the server by a factor of six.

PDM strives not to leak information and avoids timing attacks by properly order-

ing cryptographic operations. PDM can also avoid storing password-equivalent data on the server. If the server is compromised, the user's password can remain safe. Other protocols avoid password equivalence by having extra Diffie-Hellman exchanges.

Deriving a 512-bit prime from a password is computationally expensive. Ten seconds on a reasonably modern machine is not uncommon. However, there are simple improvements. Perlman's son improved the client performance by a factor of three by using a sieve instead of division. If a user provides a hint in addition to the password, the generation of the prime can finish in a fraction of a second. The hint could be the first few bits of the prime, easily encoded as a single character to remember.

Then came questions. Asked about the distribution of primes derived from passwords, Perlman answered that the primes are uniformly distributed in the range of possible primes. For all possible passwords, this is uniformly distributed.

Asked why PDM depends on a strong Sophie Germain prime, Radia explained that the base 2 is then guaranteed to be a generator if the prime is also congruent to 3 mod 8. If 2 were not a generator, then 2 would generate a smaller subgroup – reducing security.

Detecting Steganographic Content on the Internet

Niels Provos, CITI, University of Michigan

Because Slashdot had just discussed a "theoretical" system to detect steganographic content on the Internet, Niels decided it was time to discuss a system already doing this. Instead of talking about methods to defend against statistical steganalysis, Niels talked about his software to find hidden messages in JPEG files.

The popular press claims that terrorists like Osama bin Laden use steganography. Of course, this is totally unsubstantiated. Hence, Niels sought answers to three questions:

- How to automatically detect steganographic content
- How to find a source of images with potentially steganographic content
- How to determine whether an image contains hidden content

Steganography is the art and science of hiding the fact that communication is happening. In modern steganography, one should only be able to detect the presence of hidden information by knowing a secret key. The goal of an adversary is to detect steganography, not necessarily to recover the message. One must select a cover medium to embed a hidden message. Bits are changed to embed a message. The original cover medium is then destroyed.


Niels Provos

There are many systems to hide messages in images: JSteg, JPHide, and Niels' Outguess. All of these systems cause different distortions in images. Niels wrote the "stegdetect" program to detect images modified by JSteg, JPHide, and Outguess. The program gives a notion of how likely it is that an image contains hidden content.

On a 1200MHz Pentium III, stegbreak processes 15,000 words/sec for JPHide, 47,000 words/sec for Outguess, and 112,000 words/sec for JSteg. Because a single fast machine can only process so much, Niels wrote the "disconcert" program to mount a distributed dictionary attack.

Niels has sorted through over 2 million JPEG images from eBay. Although 17,000 images came up positive, no genuine steganography was found. There is as yet no final conclusion on whether the underworld uses steganography in this way. The popular press will have to continue with unsubstantiated claims.

Asked if one can determine the quality metric used to create a JPEG, Niels said this is possible but will not reveal whether there is steganographic content because modifications of DCT coefficients do not modify quality of images much.

Another person asked for advice on how to hide messages while minimizing distortion. Niels explained that hiding just one bit is easy. Otherwise it is important to realign the statistical properties of the image after embedding a message.

One audience member suggested that terrorists might use homebrew steganographic software. In such a case, will the same statistical tests help detect hidden messages? Niels said that with certain generic assumptions, maybe. One would need to know the statistical signature common to the software.

Another audience member asked if Niels has searched for JPEGs on sites other than eBay. Niels responded that he has only considered eBay because the popular press mentioned auctions as the perfect venue. So far the press seems to be fantasizing.

Finally, a participant asked if the number of false positives fit any hypothesis. Niels answered no. The images vary in quality and size. So, from the beginning, many images are mischaracterized by the statistical tests. Niels did run his software against a test set though. It correctly detected the hidden messages.

For more information, see *http://www.citi.umich.edu/u/provos/* or *http://www.outguess.org/*.

TIMING ANALYSIS OF KEYSTROKES AND TIMING ATTACKS ON SSH

Dawn Xiaodong Song, David Wagner, and Xuqing Tian, University of California, Berkeley

Dawn Song explained how two traffic analysis vulnerabilities in the SSH protocol can leak damaging amounts of information. By eavesdropping on an SSH session, Song demonstrated the ease of recovering confidential data such as root passwords typed over an SSH connection. Song's group then built the Herbivore attacker system, which tries to learn users' passwords by monitoring SSH sessions. Herbivore can speed up brute force password searches by a factor of 50.

The SSH protocol has largely replaced insecure Telnet. Ideally SSH should withstand attacks by eavesdroppers. Alas, SSH leaks information about the approximate length of data. Moreover, each key press generates a separate packet. The length can indicate when a user is about to enter a password during an established SSH session. By watching the inter-keystroke events, an eavesdropper can make educated guesses about passwords and other confidential information.

The most startling example is that of the su command typed over an SSH session, which results in a very recognizable traffic signature. Simply by looking at the lengths of requests and responses, an eavesdropper can detect the transmission of a password. Song noted that su disables echo mode. The resulting asymmetric traffic indicates that a password will follow.

Once an eavesdropper knows that a sequence of packets corresponds to a password, the inter-keystroke timings can reveal characteristics of the password. Herbivore looks at the frequency distribution of a given character pair. For instance, one may type vo with alternating hands while typing vb with the same hand. The latency between each keypress is distinguishing. For randomly chosen passwords, inter-keystroke timings leak about 1.2 bits per character.

One countermeasure against this attack would be to hide inter-keystroke timings by using a constant packet rate in active traffic.

Next, a slew of people raced to the microphone. One person asked whether taking many samples of a single user would reduce the password search space even more. Song responded that this technique has diminishing returns.

Asked about the effect this work has on passwords typed over a wireless network, Song reported that her group did not test real users' passwords. Each test subject used an assigned password. All the test subjects were touch typists.

When one audience member asked why not set TCPNODELAY right before typing passwords, another audience member said that is already the case.

Song also explained that randomly inserting a delay in traffic will not help much. An eavesdropper can obtain your typing of passwords many times to filter out the randomization.

WORKS IN PROGRESS

Summarized by Sam Weiler and David Richard Larochelle

USING THE FLUHRER, MANTIN, AND SHAMIR ATTACK TO BREAK WEP

Adam Stubblefield, Rice University; John Ioannidis and Avi Rubin, AT&T Research

The authors implemented a recently published attack against WEP, the link-layer security protocol for 802.11 networks. Exploiting WEP's improper use of RC4 initialization vectors, they recovered a 128-bit key from a production network using a passive attack. For assorted legal and moral reasons, they're not planning to release the code, but others are developing similar tools.

For more information, visit *http://www.cs.rice.edu/~astubble/wep/*.

SRMAIL – THE SECURE REMAILER

Cory Cohen, CERT

SRMail allows groups of people who may not share common crypto methods to communicate. It can generate encrypted form letters and convert between encryption formats when used as a remailer. SRMail will be used at CERT to allow several people to masquerade as CERT and generate documents signed with CERT's keys without requiring them to have direct access to those keys.

VOMIT – VOICE OVER MISCONFIGURED INTERNET TELEPHONES

Niels Provos, CITI, University of Michigan

Vomit converts a Cisco IP phone conversation into a wave file, allowing users to play a call directly from the network or from a tcpdump output file. Vomit can also insert wave files into ongoing telephone conversations. Provos suggested that Vomit can be used as a network debugging tool, a speaker phone, and so on.

For more information, visit *http://www.monkey.org/~provos/vomit/*.

VILLAIN-TO-VICTIM (V2V) PROTOCOLS, A NEW THREAT

Matthias Bauer, Institut für Informatik

Bauer amused us with several ways to transport or temporarily store data on correctly configured machines without the consent of the owner (i.e., in Web guest books, in ICMP-echo-request datagrams sent over connections with long RTTs, or in SMTP messages sent via open relays to domains that refuse to accept the messages for several days). In addition to providing an unreliable backup medium, these methods can be used to build an unobservable channel. He proposes that these theft-of-service attacks should be called "villain-to-

victim" computing because some of the engineering problems of P2P can be solved by V2V protocols.

For more information, visit *http://www1.informatik.unierlangen.de/~bauer/new/v2v.html* .

Detecting Manipulated Remote Call Streams

Jonathon Giffin, Bart Miller and Somesh Jha, University of Wisconsin

In a distributed grid computing environment, remotely executing processes send call requests back to the originating machine. A hostile user may manipulate these streams of calls. This technique statically analyzes the process's binary code at dispatch time and generates a model of all possible call sequences. As calls come back during execution, they're checked against the model, which detects some types of manipulation.

A Quantitative Analysis of Anonymous Communications

Yong Guan, Xinwen Fu, Riccardo Bettati, and Wei Zhao, Texas A&M University

This probabilistic analysis of rerouting systems found that longer paths don't necessarily provide better protection against sender identification. They also found that path complexity doesn't have a significant impact on the probability of identifying a sender. Additionally, the ease of identifying a sender increases as the number of compromised nodes in the system increases, but that growth is sublinear.

For more information, visit *http://netcamo.cs.tamu.edu/*.

Distributed Authorization with Hardware Tokens

Stefan Wieseckel and Matthias Bauer, Friedrich-Alexander-University Erlangen-Nuernberg

The authors have written a PAM module for user authentication to workstations based on RSA credentials stored on a Dallas Semiconductor Java-iButton. They use the KeyNote policy engine to make authorization decisions, which allows for complex trust relationships and delegation of authority. They do not presently address user or token revocation.

For more information, visit *http://www.wieseckel.de/ibutton_smartcard.html* .

Moving from Detection to Recovery and Analysis

George Dunlap, University of Michigan

Dunlap proposed a mechanism of rollback and selective replay of network events to aid in intrusion analysis and recovery. Being able to answer questions like "What if this packet had not been delivered?" or "What if this TCP session hadn't happened?" should facilitate debugging, forensic analysis, and intrusion detection signature development.

A Cryptanalysis of the High-Bandwidth Digital Content Protection (HDCP) System

Rob Johnson, Dawn Song, and David Wagner, University of California at Berkeley; Ian Goldberg, Zero Knowledge Systems; and Scott Crosby, Carnegie Mellon University.

HDCP is a proposed identity-based cryptosystem for use over the Digital Visual Interface bus, a consumer video bus already in widespread use. The authors found serious design flaws in HDCP which allow one to eavesdrop on HDCP communications, clone HDCP devices, and build an HDCP-compliant device that cannot be disabled via HDCP's Key Revocation facilities. Because of the DMCA mess (see page 7, the summary of "Reading Between the Lines: Lessons from the SDMI Challenge," particularly the question regarding whether a person would be at risk for summarizing the session), they aren't releasing the full details of their cryptanalysis.

Trust, Servers, and Clients

Sean Smith, Dartmouth University

WebALPS extends an SSL connection into a tamper-resistant coprocessor. By using the coprocessor as a trusted third party, sensitive information is protected from rogue server operators. Credit card information, for example, can be sent from the coprocessor via encrypted email to a merchant with the web hosting provider never having access to it.

Additionally, Smith described how SSL connections can be spoofed and presented an impressive demo in which Java script and DHTML were used to spoof the URL, the SSL warning windows, the SSL icon, and the certificate information.

For more information, visit *http://www.cs.dartmouth.edu/~pkilab.*

Source Router Approach to DDoS Defense

Jelena Mirkovic and Peter Reiher, University of California, Los Angeles

The authors propose a system to prevent a network from participating in a DDoS attack. Located at the source network router, the system watches for a drop-off in reverse traffic from a particular destination with heavy outgoing traffic. It then throttles all traffic to that destination while attempting to identify attacking flows and machines. The system is similar to MULTOPS, but its source side only, and its traffic models don't depend on packet ratios.

For more information, visit *http://fmg-www.cs.ucla.edu/ddos*.

SAVE: Source Address Validity Enforcement Protocol

Jun Li, Jelena Mirkovic, Mengqiu Wang, Peter Reiher, and Lixia Zhang, University of California, Los Angeles

SAVE is a new protocol for building incoming address tables at routers, even in the face of asymmetric routes. Those tables can be used to filter out packets with spoofed IP source addresses, build multicast trees, debug network problems, etc. To build the tables, SAVE sends valid source address information downstream along the paths used for delivery.

For more information, visit *http://fmg-www.cs.ucla.edu/adas/*.

Code Red, the Second Coming — From Whence Diurnal Cycles

Colleen Shannon and David Moore, CAIDA

Using the same system presented in the Denial of Service session on Wednesday morning, CAIDA analyzed the second round of Code Red. They observed that many of the infected hosts were using dynamic addressing, suggesting that the owners were not intentionally running IIS. The data also showed a clear diurnal pattern – one-third to one-half of infected machines were being turned on and off daily – again suggesting that these machines were not running production Web servers.

For more information, visit *http://www.caida.org/analysis/security/code-red/*.

Fast-Track Session Establishment for TLS

Hovav Shacham and Dan Boneh, Stanford University

The authors describe a new, "fast-track" handshake mechanism for TLS. A fast-track client caches a server's public parameters and certain client-server negotiated parameters in the course of an initial, enabling handshake; these need not be present on subsequent handshakes. The new mechanism reduces both network traffic and flows and requires no additional server state. The bandwidth savings are particularly relevant to wireless devices.

For more information, visit *http://crypto.stanford.edu/*.

Electromagnetic Attacks on Chip Cards

Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi, IBM Research

Chip cards and other devices leak substantially more information through electromagnetic emanations than through other side-channels such as power consumption and timing analysis. Additionally, the countermeasures for the other side-channel attacks are often insufficient to protect from electromagnetic attacks. Because of the sensitive nature of this work, the authors are working with interested parties to secure vulnerable devices prior to disclosing complete details.

For more information, visit *http://www.research.ibm.com/intsec*.

Password Authentication

Philippe Golle, Stanford University

Philippe Golle proposed a scheme for authenticating to a large number of Web sites with different passwords, while requiring the client to remember only a single master password. The scheme can be adapted to master passwords as short as 40 bits and can resist coalitions of up to three Web sites.

For more information, visit *http://crypto.stanford.edu/~pgolle*.

A Traffic Capture and Analysis Framework

Josh Gentry, Southwest Cyberport

Josh Gentry presented some Perl tools for collecting network statistics. The capture engine uses libpcap to collect traffic, does some pattern matching and analysis, and stores the results in Perl hashes. The command line client can query that data locally or over the network.

For more information, visit *http://www.systemstability.org/*.

Open Source Implementation of 802.1x

Arunesh Mishra, Maryland Information and Systems Security Lab, University of Maryland

Lib1x is an open source implementation of 802.1x, a port-based authentication mechanism for wireless networks that's intended to be an alternative to 802.11 WEP (see the WiP by Adam Stubblefield et al., above, for details on why an alternative is needed). Contributions are welcomed.

For more information, visit *http://www.missl.cs.umd.edu/1x/*.

[Photographs of the Symposium can be found at *http://www.usenix.org/events/sec01/index.html*]

Will the real Peter Honeyman please stand up!

incident response:

Performing Investigations on a Live Host

by Keith J. Jones

Keith Jones is a computer forensic consultant for Foundstone. His primary area of concentration is incident response program development and computer forensics.

Keith.jones@foundstone.com

Corporate IT staffs are investigating computer security incidents and computer crime more than ever before. Who would have thought the IT staff would become the "network cops" of the company? But that is exactly what they have become. Therefore, your Incident Response (IR) staff needs to be armed and prepared to support the decisions and investigations to protect corporate assets, protect employee privacy, and enforce the policies that general counsel and senior management endorse. A methodology and formal investigative process needs to be implemented.

This article will describe the process of performing a successful live incident response on a UNIX operating system and will discuss the mechanisms used to preserve the evidence. It is assumed the reader has basic system administration skills and little or no experience with investigations. Therefore, this article will provide deeper focus on the investigative aspects of the live response and methods used to collect the evidence in a forensically sound manner rather than the technical usage of the tools. Since no investigation is the same, a step-by-step process that will encompass every aspect you may encounter is difficult to provide. The information in this article will provide a solid base to executing and transferring most of the information needed for a successful investigation in a forensically sound manner.

Choosing the Toolkit for the Investigation

In order for the investigator to examine the victim machine, the correct tools must be used. To select the tools you use, keep the "big picture" in your mind. What log files do you want to retrieve? What UNIX tools help you determine the state of the system? What configuration files do you want to review? Before you create your trusted toolkit, you must determine the information you wish to acquire. The tools used in the live investigation of computer crimes are not specialized. They resemble a handful of items in the typical system administrator's toolbox. Minimally, the following areas of a suspect machine need to be examined during an investigation:

1. The current date and time of the victim server are recorded. This information will provide a baseline if the system time has been skewed from other servers present in the investigation.

   - Tool: *date*

2. All information about each network interface card, such as network addresses and states, is recorded. If the suspect altered the IP address or started a network monitoring tool (e.g., sniffer) on the victim machine, this data would display the mischievous action.

   - Tool: ifconfig
   - Typical Switches: -a

3. Each name, command line argument, length of execution time, and the user who executed the process are recorded. Rogue processes are typically initiated by intruders, and the currently running process will be part of the evidence used to prove that the rogue process was executed.

- Tool: ps
  - Typical Switches: -aux (or –ef for Solaris)

4. The open network sockets and files along with the process number that opened them are recorded. Typically, intruders leave back doors for future external access into compromised systems. External access will require network access, and open sockets are the indicator of a back door's presence. Furthermore, unknown open files usually indicate a monitoring tool that is either reading or writing to a file. As an example, the most common type of monitoring tool is a sniffer, and it writes to a log file.

- Tool: netstat
  - Typical Switches: -an
- Tool: lsof

5. Since an executable file can be deleted from the file system and currently be executing, your initial response may be the only chance to capture a copy and perform offline tool analysis on a rogue process. All suspect processes are archived in a forensic manner as an executable file.

- Tool: the proc file system
  - Executable Image Location: /proc/<PID>/exe
- Tool: carbonite (for Linux)
  - Location: *http://www.foundstone.com*

6. The IP routing table is documented. Unauthorized routing of the victim machine's network packets will be evident in the routing table, and it may indicate possible man-in-the-middle network monitoring and attacks.

- Tool: netstat
  - Typical Switches: -nr

7. The currently logged in users and their initiating place of connection are documented.

- Tool: w

8. The loaded kernel modules (if your version of UNIX allows for this) are documented. Kernel modules literally alter the operating system and are in the intruder's benefit to load them, obviously an area that needs to be examined by the investigator.

- Tool: lsmod

9. The last accessed, modified, and created timestamp information for each file on the victim machine are recorded. Correlating the timestamps for significant events provides information such as last execution, unauthorized modifications, and other mischief. Furthermore, file permissions play a significant role in most investigations and are easily captured during the same step as timestamps.

- Tool: ls
  - Typical Switches for Last Accessed Time: -alR –time=atime /
  - Typical Switches for Last Modified Time: -alR /
  - Typical Switches for Created Time: -alR –time=ctime /
- Tool: find

. . . your initial response may be the only chance to capture a copy and perform offline tool analysis on a rogue process

The simplest back door installed by an intruder is a shell, such as bash, listening on a random port.

- Tool: The Coroner's Toolkit (mactime)
- Location: *http://www.porcupine.org*

10. All auditing files are archived into evidence. UNIX auditing is typically performed by the syslog facility, and each log file is determined from the /etc/syslog.conf file. In addition to these logs, the wtmp, utmp, and lastlog files are archived into evidence so that the login history is available for the investigation.

- Tool: cat
- Tool: last

11. The /etc/passwd file is submitted into evidence. This file will be examined for possible back doors into the system. It is well known that valid, but not necessarily authorized, user credentials are the easiest way to avoid intrusion detection systems and simple access.

- Tool: cat

12. The /etc/inetd.conf file is submitted into evidence. The simplest back door installed by an intruder is a shell, such as bash, listening on a random port. This back door will be easily observed in this configuration file.

- Tool: cat

13. All suspicious files on the victim machine are submitted into evidence for offline tool analysis. The files can be transferred by using cat and redirecting the output to a netcat TCP session, which is explained in the next session.

- Tool: cat
- Tool: strings
- Tool: strace
- Tool: file

It has to be assumed that any tool resident on the victim machine may be compromised. If the intruder used a publicly available rootkit, they can trojan any system tool such as ps, netstat, ls, and even bash to provide false results to the user executing them. The tools listed above must be executed within a trusted shell, which is compiled on a system without a history of incidents. Therefore, it is necessary to execute a trusted shell before your initial response begins. Furthermore, the tools you use must be compiled statically on a forensic workstation and transferred to the victim machine. By statically compiling your tool set, you avoid running untrusted, potentially damaging processes on the victim machine. The tools can be accessed by writing them either to floppy or CD-ROM. This toolkit media is inserted into the victim machine and mounted. Additionally, it is highly recommended to perform the response from the victim console and not an X-Windows session. There are security considerations such as session spying inherent with X-Windows usage. Furthermore, a response should never be performed across a network connection such as Telnet. Once the media is mounted, the trusted shell is run by executing it on the command line. The response can begin once the investigator has completed the proper documentation and planning steps, explained in an upcoming section.

Storage of the Digital Evidence

There are several options for storing the data produced in the last section. The first and most intrusive method is to save the data on the victim machine's hard disk for

analysis. The second method, less obtrusive but also less practical, is to store the data on external media such as a floppy disk. The third and least obtrusive method involves transmitting the data to a forensic analysis machine and saving on its hard disk drive. This is the method that is recommended and is used in the rest of this article.

The method of saving the data directly to the forensic machine uses a TCP/IP network as the transmission medium. The tool that will easily establish a TCP session is named netcat. netcat is used in two modes: connection mode or server mode. The victim machine will utilize the connection mode while the forensic analysis machine will use the server mode. Since netcat establishes a TCP session, information can be sent through the connection to a server by using the command line pipe. It is possible to transfer whole files from one machine to another by simply redirecting the data on the forensic machine that was received through netcat. The complete transmission process can be summed up with the following commands:

Victim Machine: <command line> | nc <IP of the forensic workstation> <port number>

Forensic Workstation: nc –l –p <port number> > <command line>.txt

Not every network used to transfer the data from the response will be trusted. To overcome this hurdle, another tool named cryptcat can be used in the same manner as netcat. The difference between the tools is that cryptcat encrypts the TCP channel. The encryption provides two aspects: authenticity and secrecy. If a bit were changed in transmission, the data would be invalid when received on the forensic machine. Additionally, if an intruder is listening with a network monitoring tool, he or she will observe garbled data due to the encryption.

There are two realistic choices for the network transfer of incident data: create a temporary network using a crossover cable between the forensic workstation and the victim machine or connect the forensic machine directly to the untrusted network. A topic of debate is whether the responder chooses to remove the victim machine from the live, untrusted network at the time of detection. One logical approach to this question is to perform the smallest subset of investigative steps to find out if removing it from the live, untrusted network will trigger malicious code the intruder left behind. Additionally, leaving the victim machine on the network will give the investigator the chance to collect evidence if the intruder were to return by passively monitoring the situation with a network monitoring tool.

Since every investigation should be performed under the pretense that it is the "big one," it is assumed that one day the investigator will be called to the stand to swear under oath that the data is unaltered. There will be a mechanism in place to validate the authenticity of the data at any point if it is questioned. The mechanism generally accepted by the industry is an MD5 checksum. The MD5 checksum is a 128-bit length string computed from a file's contents and it is highly unlikely that two files will have the same value. To create an MD5 checksum list of several files, the following command will work well:

Forensic Workstation: md5sum –b <filenames> > md5sums.txt

The MD5 checksum will be computed for every file transferred from the victim machine to the forensic workstation. The checksum will be computed and saved on the forensic server itself. Lastly, the evidence data and the MD5 checksum file will be copied to unalterable media such as CD-ROM with the disc closed after the write.

Not every network used to transfer the data from the response will be trusted.

Documentation and Planning

Typically, documentation does not come naturally to technical individuals. However, documenting the steps taken during an incident response is paramount. Records of a response performed months or years prior have a longer shelf life than an individual's memory. Planning is also very important to the response because sometimes the investigator may only have one chance to respond correctly. Planning the commands, the order, and what switches will be used on the victim machine will follow hand-in-hand with the documentation. A simple spreadsheet is used to document what commands executed on the victim machine can be created before the response is performed, therefore allowing the investigator to plan and research the tools before they are run live during the response. An example of this spreadsheet is viewed below.

Start Time	Command Line	Trusted Execution	Untrusted Execution	md5sum	Comments
4/3/2001 10:37:47	date | nc 192.168.69.2 2222	X		e913412389f3430c5662a3ee54aef082	daylight savings time in effect.
4/3/2001 10:42:15	netstat –an | nc 192.168.69.2 2222	X		37cfdab36f8e42369f099d39af36b275	

The columns "trusted execution" and "untrusted execution" indicate how the tool was executed and are the proper place for this documentation. In these columns it will be considered an untrusted execution if any code contained on the victim machine is run, such as dynamically loaded libraries and other tools.

Another aspect that must be documented is the transfer of evidence. The chain of custody is the record of when, to whom, and where a piece of evidence is transferred. A completed chain of custody form will provide an extra level of authenticity of the evidence, if it is ever questioned, and is standard for any law enforcement investigation. A sample chain-of-custody form is observed here.

Case Number: FS-010101 Evidence Tag: 001		**EVIDENCE** CHAIN OF CUSTODY			
Evidence Description:	CD-ROM containing live response data files				
Source Location:	X	Source Name:	X	Date:	06/01/2001 12:10
Destination Location:	Onsite Investigation Miami, FL.	Destination Name:	Keith J. Jones	<Keith's Signature>	
Source Location:	Washington, DC	Source Name:	Keith J. Jones	Date:	06/02/2001 14:43
Destination Location:	Evidence Safe, Washington, DC	Destination Name:	Keith J. Jones	<Keith's Signature>	
Source Location:	Washington, DC	Source Name:	Keith J. Jones	Date:	06/04/2001 15:33
Destination Location:	Washington, DC	Destination Name:	Kevin Mandia	<Keith's Signature>	<Kevin's Signature>
Source Location:		Source Name:		Date:	
Destination Location:		Destination Name:			
Source Location:		Source Name:		Date:	
Destination Location:		Destination Name:			

If the proper documentation is created throughout the investigation, a final report is simple to compile. The information can be summarized from the various documentation sources easily. Simple generation of the final report can be the greatest motivation to document properly.

Conclusion

This article provides a summary of tools and techniques used for a successful live incident response on a UNIX operating system. Proper documentation and planning are needed in order to keep mistakes to a minimum. Documentation includes documenting the chain of custody to uphold the authenticity of the evidence should legal action follow an investigation. A toolkit of trusted binary files must be used during the response. The toolkit should be compiled statically and transferred to the victim machine before the response is executed. Lastly, steps such as calculating an MD5 checksum on the evidence and archiving it to a read-only media will also be used to prove authenticity.

These principles can be easily applied to a Microsoft Windows NT/2000 machine. Substitution of the UNIX tools with ones that are similar for NT/2000 will need to be performed, but that is not difficult. The documentation and planning stages will be exactly the same.

Proper documentation and planning are needed in order to keep mistakes to a minimum.

forensics lite

by Brad Powell

Brad Powell works as a senior network security consultant for Sun Microsystems Professional Services, where he does security research and writes security tools.

brad.powell@sun.com

As the old Boy Scout motto goes, "Be Prepared." Preparation is the key to successfully surviving a security incident. If your organization hasn't thought about how you are going to handle that future security incident, then NOW is the time to start thinking about it.

Let's discuss some reasons why. When I was working internal security for Sun, we figured out four or five likely security-incident scenarios and laid plans on how they would be handled and coordinated. The plans didn't all survive the real-life incidents, and some ended up being changed on the fly to deal with things we didn't expect, but all in all, they allowed us to bring the right resources to the problem. Having a plan gave us the ability to work through all the surprises while still keeping our sanity – a good reason to have one in this day and age.

Hoever, in my experience, most sites are not prepared and don't have a plan in place, so we will address that scenario in detail.

How to Respond to an Attack

So you think you have been hacked and don't know how to respond. First off, don't PANIC. I've handled plenty of intrusions and a large percentage of them are false alarms. Secondly, by the time you have figured out that you truly have been hacked, it's probably too late to panic, so I'd advise skipping the panic step altogether.

So what do you do? Well, the course is often dictated to you by management. Your chances of catching the intruder are pretty small, so you might ask yourself "Why bother?" Management, who may be on your case to get the resource back up and running in a hurry, doesn't want to spend the time going through a full forensic analysis; all they want is that Web server "Back on the Net so we can conduct business." If this is an all too familiar scenario, I beg you, when you get done reading this article, take it to your management and have them read it too. Things will never get better until your organization starts prosecuting and handling incidents.

We seem to have two opposing problems. Management wants and needs to get the system back up and running quickly, but doing forensic analysis takes a lot of time. I submit to you that just getting the system back up on the Net begs the attacker to hit you again and again. Until you fix the problems, you're living on borrowed time.

Here is an example. When working internal security for Sun Microsystems, we had an incident involving Kevin Mitnick. It was only because of a Herculean effort by Tsutomu Shimomura that Mitnick was captured, but it was because Sun (and others) collected so much evidence that Mitnick pleaded guilty to the charges. The point is that the evidence you collect may not be readily used by your site, but it still has value. My copy of the evidence as well as the original disk drives sat in a safe gathering dust for over three years before the evidence was ever seen in court.

All well and good, but what if you don't know if the system has been compromised? Well your Network IDS and local Tripwire data will tell you what has occurred and what has changed. What? You don't have either of those? I'm not surprised. Now you have a decision to make. Again, it's probably based on resources. At this point you can:

- Call in a professional to examine the tampered disk and give you an analysis of what went wrong.
- Turn over the evidence to law enforcement.
- Look at the disk yourself and see what you can learn from it.

- Forget the whole thing.

Sadly, the last option is taken all too often. I beg you to reconsider before taking this option.

Since we are discussing "forensics lite," I'll leave the first two options for another article, or for others to cover, and will focus on what you can learn from the information to ensure that you really have plugged the hole, so you don't get any repeat incidents, and to ensure you have cleaned up the mess.

Our first step is to make sure we really have been hacked. This can be trickier than you think.

Our first step is to make sure we really have been hacked. This can be trickier than you think. Why? Well, to do it right, to avoid tampering with the evidence, you have to follow all the procedures as if you are sure you have been hacked. So we assume we have been hacked even though we are still unsure. First, take the system offline. Then, to preserve the memory and running process tables, we need to suspend the system without doing a graceful shutdown using shutdown(1) or init 0. With a Solaris system, we use the Stop-A or L1-A sequence. Check with your operating system vendor for equivalent functionality. Then we mount the file system onto another system, make a full backup, preserve the original disk if at all possible, and build a new system disk that is hardened against attack. Do NOT reuse the system backup you just made to restore the system, since that is evidence, and most likely will only reintroduce the old bug back into the system if used. That system cannot be trusted.

This is the safest way, and even if you find out later that you have not been hacked and discover this was a false alarm, you have still improved your system security. Consider the fresh install an added bonus and a worthy exercise.

You may later have to reload portions of the suspect system backup, but what gets reloaded should not be the operating system or any other component that you can get from distribution media. Make sure to apply all the latest patches after rebuilding from distribution media.

The backup and fresh install gets the organization back up on the network and conducting business and gives you the opportunity to figure out what went wrong and how you can prevent it in the future. It also gives you, if you do discover evidence that may lead to prosecution, a clean copy of all evidence on the original system disk and on a backup tape.

From here on, I'm assuming we are working with a copy of the data and not with the original. For full details on doing full forensics and how to make a copy of the disk using dd(1) for your forensic lite, refer to TCT, The Coroner's Toolkit (*http://www.fish.com/tct/*), or "DD and Computer Forensics," a simple guide to using the dd(1) command, by Thomas Rude (*http://www.crazytrain.com/dd.html*). We use the UNIX dd(1M) command instead of using tar(1) or ufsdump(1M) so that we preserve the byte-by-byte layout of the system disk and the swap area of the disk. This is important if we decide later that we need to do more in-depth forensics using TCT or another forensic tool.

Now that we have a working copy and have preserved the original, there are some simple things we can do to poke at the disk to see if the operating system has been modified. The most often seen problem is a rootkit. A rootkit is a replacement set of Trojan binaries that is used by intruders to hide their use of the system and used to install back-door access to your system for future use.

Since we cannot assume the disk we are looking at hasn't been trojaned or rootkitted, we don't boot it

Again, since we are talking about a lite version of forensics, I'm going to stick with basic and simple tools. Let's start with Sidekick. Sidekick.sh is a simple Bourne shell script that works with MD5 to check the signatures of files. Sidekick is available from *http://www.sun.com/security/* or from *http://www.sun.com/blueprints/tools/fingerprint_license.html*. Although written for use with the Solaris Operating System, sidekick.sh is portable enough to use on most any UNIX-like operating system. We use Sidekick in this example instead of going to an industry standard like Tripwire because Tripwire can only report modifications since the last time you ran it. If you had run Tripwire before all this occurred, then you would already know what has changed. You might want to think about using Tripwire in the future (it is a really good tool/product), but without the previous baseline to compare against, you can't figure out what has changed. So we use Sidekick to try and create this baseline. As a side note, Sidekick can be used to look at legacy systems that you can't take offline to verify that the system hasn't been trojaned in the past; and from that point on, Tripwire ran daily will give you a reasonable start at system-level intrusion detection. This isn't a full solution, just a start.

Sidekick.sh, as the name implies, is a simple companion tool that just aids you in the collection of MD5 signatures. When used on a file, MD5 produces a cryptographic checksum of the file. It is nearly impossible for two binary files (assuming their length is not 0) to produce the same MD5 checksum unless the content of the binaries match. Thus, if we have two files with identical names but different contents, the MD5 checksums will not match. Please read the Sidekick manual page that accompanies distribution of sidekick.sh for details. I worked hard to make sure the man page had useful information and limitations on its effectiveness.

Since we cannot assume the disk we are looking at hasn't been trojaned or rootkitted we don't boot it; instead, we mount the partitions and gather the MD5 checksums from there, or in the case of a system we can't take offline, we run sidekick.sh locally on the system using a statically linked MD5 program and take the resulting MD5 checksum output to another non-suspect system and check the MD5 checksums from there.

Some of the useful options for sidekick.sh include:

-R Specify an alternate root directory. This option is useful for chroot areas such as on DMZ Web servers, and can be used when you are able to mount the various partitions of the suspect disk onto another system.

-r Check all the files that are commonly rootkitted, comparing them against what Sun ships. This is the basic common Trojan check and should be performed before any other Sidekick option to verify that the find(1) command, at a minimum, hasn't been trojaned.

-S This option is used with any of the other options to run Sidekick standalone, that is, without invoking the Perl script that compares checksums (explained below).

-a This option checks all files and creates the MD5 checksum. This option is a poor man's version of L5 or Tripwire and can be used to check a legacy system and create a reasonable baseline to ensure the binaries that are currently on the system are the ones shipped by Sun for the entire system. Once a baseline is determined, the user can then run L5 or Tripwire on a regular basis looking for changes. See the warning above.

Caution should be taken when using this option, as there will be many false positives reported for any locally modified configuration files or binaries.

So a typical check might include running sidekick.sh -R /mnt/suspectdisk -a -S which collects an MD5 checksum of all the system files started at the mount point. sidekick.sh -r just collects the MD5 checksums of a subset of binaries that are commonly found in a rootkit.

Now that we have MD5 checksums of all the files, we can compare this list to a list of MD5 checksums from a known uncompromised system. Whatever doesn't match is either a patched binary that didn't come with the original vendor release or a Trojan.

In the case of Sun Microsystems and Solaris, the task of finding a listing of known good MD5 checksums has been made easier for you. Sun provides an MD5 fingerprint database of every binary that it has ever shipped. I strongly encourage you to get with your other vendors (Linux people, are you listening?) to have them supply MD5 checksums of every binary they ship. An additional tool called sfpC.pl is a Perl script that can be used with sidekick.sh to query Sun's Fingerprint database online. More information about the Sun Fingerprint database can be found by searching on the keyword "fingerprint database" at *http://www.sun.com/blueprints/* or by going directly to the Fingerprint database link at *http://sunsolve.Sun.COM/pub-cgi/fileFingerprints.pl*.

Sun provides an MD5 fingerprint database of every binary that it has ever shipped

An example use of sidekick.sh:

```
root# sidekick.sh -S -r
Operating in standalone mode, sfpC will not be run.
Searching for files commonly found in rootkits.
The output has been saved to rootkitfiles-md5.20010820130858
[Note: the example has been edited to 10 entries to conserve space]
root# cat rootkitfiles-md5.20010820130858
MD5 (/usr/bin/du) = 9b9d5b91bb697c0b5e8acdd7e8286b79
MD5 (/usr/bin/ls) = 351f5eab0baa6eddae391f84d0a6c192
MD5 (/usr/sbin/in.telnetd) = dae9a44a49faeaf54ddcb30578663b39
MD5 (/usr/bin/login) = b6915118b96ed4132f45db7332dcc293
MD5 (/usr/sbin/route) = a4fea6e7e07e377812773c8e71cc5f05
MD5 (/usr/sbin/inetd) = 90907143eb6a4909aee4a7297b5d12a7
MD5 (/usr/bin/passwd) = 39d9e82ed48669d6396fed0fb9c0f901
MD5 (/usr/sbin/in.rshd) = 9656d16a4550c925d00e78d90cb775c9
MD5 (/usr/sbin/in.rlogind) = 46c1c2ba01e36c8264a3d25c4097bc98
MD5 (/usr/sbin/syslogd) = 93e97f044f18c85bfe3a12fb77f1198e
```

We take this output file created by sidekick.sh called rootkitfiles-md5.20010820130858 to another system which has Internet connectivity, and feed it into sfpC.pl, the utility that queries Sun's Fingerprint database:

```
mysystem-> sfpC.pl rootkitfiles-md5.20010820130858

    9b9d5b91bb697c0b5e8acdd7e8286b79 - (/usr/bin/du) - 1 match(es)

canonical-path: /usr/bin/du
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC
patch: 109803-01

351f5eab0baa6eddae391f84d0a6c192 - (/usr/bin/ls) - 1 match(es)
```

canonical-path: /usr/bin/ls
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

b6915118b96ed4132f45db7332dcc293 - (/usr/bin/login) - 1 match(es)

canonical-path: /usr/bin/login
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

a4fea6e7e07e377812773c8e71cc5f05 - (/usr/sbin/route) - 1 match(es)

canonical-path: /usr/sbin/route
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

90907143eb6a4909aee4a7297b5d12a7 - (/usr/sbin/inetd) - 1 match(es)

canonical-path: /usr/sbin/inetd
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

39d9e82ed48669d6396fed0fb9c0f901 - (/usr/bin/passwd) - 1 match(es)

canonical-path: /usr/bin/passwd
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

9656d16a4550c925d00e78d90cb775c9 - (/usr/sbin/in.rshd) - 1 match(es)

canonical-path: /usr/sbin/in.rshd
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC
patch: 108985-02

46c1c2ba01e36c8264a3d25c4097bc98 - (/usr/sbin/in.rlogind) - 1 match(es)

canonical-path: /usr/sbin/in.rlogind
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

93e97f044f18c85bfe3a12fb77f1198e - (/usr/sbin/syslogd) - 1 match(es)

canonical-path: /usr/sbin/syslogd
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

Now we need to examine the output from the Fingerprint database query. We notice that we received nine responses from the database; each response showed exactly one

match, a reference of which file matched the files in the database, the OS release that match came from, and in some cases, the fact that some of the binaries were Sun patches and not directly tied to a distribution.

But wait! We sent the query 10 MD5 checksums. ONE didn't receive a return; in.telnetd was not found in Sun's Fingerprint database. What does this mean? This means that the in.telnetd on the system was never shipped by Sun. Is this a Trojan? Possibly. It could also mean that the user replaced in.telnetd with another version, possibly to add a feature that Sun doesn't provide, but either way in.telnetd needs to be looked at very closely to decide where it came from and if it is legitimate.

We have now looked at all the shipped binaries for a Solaris OS, but what if I am using Linux, or FreeBSD? Well some of the same tactics apply; it is just going to take additional steps. There are a few options. First, if we have another system which was installed using the same installation media and hasn't been modified too greatly by the user (not likely), we could build up our own MD5 database. A better option would be to install a fresh system using the installation media, install any updated RPMs (patches) that were previously applied (you do keep a log of all patches you have applied to each system, don't you?) then use it as root so we get every file:

```
root-on-clean-system# find / -type f -exec md5 {} \; >clean-md5db-ostype-date
```

Then you sort and unique the files to give an alphabetic listing of all files:

```
root-on-clean-system# sort -u clean-md5db-ostype-date >
            clean-md5db-ostype-date-sorted
```

Then we can do a simple compare of the two files, the MD5s collected from our suspect system with sidekick.sh -a and the one from a clean system.

```
root-on-suspect-system# sidekick.sh -S -a
Operating in standalone mode, sfpC will not be run.
Searching for all files commonly this option is used in conjunction with '-S'
The output has been saved to allfiles-md5
```

We then move or copy the allfiles-md5 to the clean system; run sort and unique there, and are ready to compare:

```
root-on-clean-system# sort -u allfiles-md5 >allfiles-md5-suspect
root-on-clean-system# diff allfiles-md5-suspect clean-md5db-ostype-
        date-sorted
```

This will give us a reference pointer and a starting point. This method will result in a lot of false negatives since system files such as the password file as well as any system files that have been modified will not match the MD5, but it will at least show us something meaningful.

Even though we are only doing the lite version of forensics, this is all getting complicated, isn't it? We could have saved ourselves a lot of grief up to this point if we had created an MD5 database as soon as we installed the system, and/or run something like Tripwire. An ounce of prevention is worth a pound of cure; or in this case, 20 minutes of MD5 up-front is worth 20 hours of MD5 later.

Have I scared you enough to get you to go run MD5 now before an intrusion? Good!

Even though we are only doing the lite version of forensics, this is all getting complicated, isn't it?

The first thing any intruder does is clean out their entries from the log files to mask or cloak their presence

Log Files

What about log files? Well they are worth reviewing, and if you have one local copy and one remote copy on your syslog server, then it should be a simple matter of comparing the log sizes to determine if your log files have shrunk in size. The first thing any intruder does is clean out their entries from the log files to mask or cloak their presence. Most rootkits have a built-in utility to do this. What? You only keep the log files locally and don't have a syslog server?

You are screwed. But, look at the log files anyway. Maybe it's a script kiddie, and their script failed to work properly due to a bad path to a utility or something, but don't bet on it. If you do have a syslog server set up, you are looking for anything in the remote file that doesn't match the local copy, and from there looking both before and after that entry for signs of the first penetration. Most intruders will have set up a back door that doesn't log their connections, so you need to see if there was any previous system scan or failed attempt before they found that bug in your application and introduced the rootkit into your system. This may or may not be worth the trouble. Lesson learned.

From here we need to look for anything which might have been loaded into the kernel as a module. The process table and /proc file system if your system has one (most modern UNIX systems do) are worth examining. Why? Well, after breaking into a system a clever intruder will install his or her back door and then delete their toolkit to keep it from being captured or examined. If you're lucky and have the time to do full forensics, it is quite possible to recover these deleted files. Take a look at the examples (shameless plug alert) from the Honeynet project (*http://project.honeynet.org*) or The Coroner's Toolkit. Recovery of files takes a long time and requires plenty of disk space to attempt, with no guarantee of results, but it is amazing just how pervasive data is and how hard it is to truly delete.

This may be too extensive for forensics lite, so we will try looking into the /proc file system first before attempting this.

From the man pages on proc:

> "/proc is a file system that provides access to the state of each process and lightweight process (lwp) in the system. The name of each entry in the /proc directory is a decimal number corresponding to a process ID. These entries are themselves subdirectories. Access to process state is provided by additional files contained within each subdirectory; the hierarchy is described more completely below. In this document, '/proc file' refers to a non-directory file within the hierarchy rooted at /proc. The owner of each /proc file and subdirectory is determined by the user-ID of the process."

We can use this to our advantage. Assuming we were able to suspend the system and didn't have to do a full shutdown (above), the /proc file system will be intact on our suspect disk. We can first figure out if the process running was part of the original operating system using MD5 and Sun's Fingerprint database (or the database we created using distribution media above). So let's use MD5 to gather up signatures from the running binaries from the /proc file system:

```
mysystem-> md5 /mounted-suspect-disk/proc/*/object/* >proc-md5-sigs
```

This will give us a list of MD5 signatures that we can then examine using the Fingerprint database. This step weeds out binaries that are system binaries and allows us to figure out what was running.

As an example, I take two of the MD5 signatures from processes that were running, and check it against known fingerprints:

```
mysystem-> cat proc-md5-sigs
    [edited to only have a few entries for space]
    MD5 (/proc/22296/object/a.out) = 1a7f2e29e1ee6fb0f5e87d0ba2d8770e
    MD5 (/proc/2332/object/a.out) = 2805a09d9790e70ab0e098f337603656
    MD5 (/proc/2332/object/ufs.32.6.357638) = e725b71635a1c5f1f72eff24cd3e63cc
    MD5 (/proc/2332/object/ufs.32.6.65275) = 84ab84fa8af00324e09627912178c4a0
    MD5 (/proc/2332/object/ufs.32.6.8182) = d60917907d88e9e56251ce0989eebfd4
    MD5 (/proc/2332/object/ufs.32.6.8576) = 422073158f2775bdf119c3847d0d5b64
    MD5 (/proc/2332/object/ufs.32.6.8581) = a11071db878fe24f9e03fb0b53c67bf8
    MD5 (/proc/23418/object/a.out) = 2805a09d9790e70ab0e098f337603656
    MD5 (/proc/9322/object/a.out) = f7f70ef41fdab1b7670582e62270e76c
    MD5 (/proc/23418/object/ufs.32.6.357638) =e725b71635a1c5f1f72eff24cd3e63cc
    MD5 (/proc/23419/object/a.out) = e1ee9f994caeb485767e225f049b3059

mysystem-> sfpC.pl proc-md5-sigs

    1a7f2e29e1ee6fb0f5e87d0ba2d8770e - (/proc/22296/object/a.out) - 0 match(es)

Not found in this database.

    2805a09d9790e70ab0e098f337603656 - (/proc/2332/object/a.out) - 1 match(es)

    canonical-path: /usr/bin/jsh
    package: SUNWcsu
    version: 11.8.0,REV=2000.01.08.18.12
    architecture: sparc
    source: Solaris 8/SPARC
    patch: 109324-01

    e725b71635a1c5f1f72eff24cd3e63cc - (/proc/2332/object/ufs.32.6.357638) - 1 match(es)

    canonical-path: /usr/platform/sun4u/lib/libc_psr.so.1
    package: SUNWkvm
    version: 11.8.0,REV=2000.01.08.18.12
    architecture: sparc.sun4u
    source: Solaris 8/SPARC

    84ab84fa8af00324e09627912178c4a0 - (/proc/2332/object/ufs.32.6.65275) - 1 match(es)

    canonical-path: /usr/lib/locale/en_US.ISO8859-1/en_US.ISO8859-1.so.2
    package: SUNWnamos
    version: 1.0,REV=1999.11.23.15.16
    architecture: sparc
    source: Solaris 8/SPARC

    d60917907d88e9e56251ce0989eebfd4 - (/proc/2332/object/ufs.32.6.8182) - 2 match(es)

    canonical-path: /etc/lib/ld.so.1
    package: SUNWcsr
    version: 11.8.0,REV=2000.01.08.18.12
    architecture: sparc
    source: Solaris 8/SPARC
    patch: 109147-06

    canonical-path: /usr/lib/ld.so.1
    package: SUNWcsu
    version: 11.8.0,REV=2000.01.08.18.12
    architecture: sparc
```

```
source: Solaris 8/SPARC
patch: 109147-06

422073158f2775bdf119c3847d0d5b64 - (/proc/2332/object/ufs.32.6.8576) - 1 match(es)

canonical-path: /usr/lib/libdl.so.1
package: SUNWcsl
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC
patch: 109147-06

a11071db878fe24f9e03fb0b53c67bf8 - (/proc/2332/object/ufs.32.6.8581) - 1 match(es)

canonical-path: /usr/lib/libgen.so.1
package: SUNWcsl
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

e1ee9f994caeb485767e225f049b3059 - (/proc/23419/object/a.out) - 1 match(es)

canonical-path: /usr/dt/bin/dtpad
package: SUNWdtdst
version: 1.4,REV=10.1999.12.02
architecture: sparc
source: Solaris 8/SPARC
```

Ah! We now know that process /proc/2332/object/a.out was actually /usr/bin/jsh, and we also know that process MD5 (/proc/9322/object/a.out) was not found in the database. This file needs to be examined in more detail. Anything that doesn't match will need additional work to discover what it is. A first check would be to use the what(1) command and then the strings(1) command to see if we can figure out what the file is.

```
mysystem-> what /mnt-suspect-system/proc/ 9322/object/a.out
     stream.h    1.85   99/12/15 SMI
     isa_defs.h  1.20   99/05/04 SMI
     vnode.h 1.85   99/07/30 SMI
     types.h  1.66   00/02/14 SMI
     feature_tests.h    1.18   99/07/26 SMI
     machtypes.h  1.13   99/05/04 SMI
     int_types.h1.697/08/20 SMI
     select.h 1.16   98/04/27 SMI
     time.h    2.64  99/10/05 SMI
     time.h    1.39  99/08/10 SMI
     time_iso.h 1.199/08/09 SMI
     time_impl.h    1.599/10/05 SMI
     t_lock.h 1.45   98/02/01 SMI
     machlock.h1.21   00/04/27 SMI
     param.h 1.76   00/02/14 SMI
     unistd.h 1.37   98/10/28 SMI
     mutex.h 1.20   98/02/01 SMI
     rwlock.h    1.998/02/18 SMI
     semaphore.h  1.598/02/01 SMI
     condvar.h   1.11   00/03/05 SMI
     cred.h    1.21   97/01/09 SMI
     uio.h 1.29   97/06/29 SMI
     resource.h 1.25   98/06/30 SMI
     seg_enum.h   1.395/12/22 SMI
     poll.h1.28   98/11/23 SMI
     strmdep.h  1.10   98/01/06 SMI
```

```
model.h 1.20   97/09/22 SMI
strft.h    1.1 99/07/30 SMI
dlpi.h 1.23   98/04/28 SMI
bufmod.h   1.9 98/01/06 SMI
types32.h  1.4 98/02/13 SMI
stdio.h    1.78   99/12/08 SMI
stdio_iso.h 1.2 99/10/25 SMI
va_list.h 1.12   99/05/04 SMI
stdio_tag.h 1.3 98/04/20 SMI
stdio_impl.h   1.8 99/06/10 SMI
ctype.h  1.33   99/08/10 SMI
ctype_iso.h    1.1 99/08/09 SMI
string.h  1.24   99/08/10 SMI
string_iso.h    1.2 99/11/09 SMI
file.h 1.60   99/08/31 SMI
stropts.h    1.48   99/08/31 SMI
conf.h    1.59   99/05/26 SMI
signal.h  1.54   99/07/26 SMI
signal_iso.h    1.1 99/08/09 SMI
siginfo.h    1.54   98/03/27 SMI
machsig.h  1.15   99/08/15 SMI
socket.h 1.53   99/11/07 SMI
netconfig.h    1.20   99/04/27 SMI
in.h    1.29   00/03/28 SMI
byteorder.h    1.14   98/04/19 SMI
un.h  1.9 96/07/12 SMI
if_dl.h    1.7 98/01/06 SMI
ioctl.h    1.9 92/07/14 SMI
if.h    1.23   00/03/28 SMI
if_arp.h  1.4 98/01/06 SMI
if_ether.h   1.8 99/03/21 SMI
in_systm.h 1.5 98/01/06 SMI
ip.h    1.7 98/08/26 SMI
udp.h 1.6 99/11/04 SMI
ip_var.h  1.4 98/01/06 SMI
udp_var.h   1.2 93/02/04 SMI
tcp.h 1.14   99/11/04 SMI
inttypes.h   1.2 98/01/16 SMI
int_limits.h 1.6 99/08/06 SMI
int_const.h 1.2 96/07/08 SMI
int_fmtio.h 1.2 96/07/08 SMI
ip_icmp.h   1.4 99/04/05 SMI
netdb.h 1.23   99/12/06 SMI
inet.h    1.17   99/03/21 SMI
```

This tells us that the binary was compiled using Solaris header files, and it also tells us that this binary probably does some networking functions based on the types of header files it uses, such as tcp.h and socket.h. Let's look at the strings output.

```
mysystem-> strings mnt-suspect-system/proc/ 9322/object/a.out
— TCP/IP LOG — TM: %s —
    PATH: %s(%s) =>
    %s(%s)
    STAT: %s, %d pkts, %d bytes [%s]
    DATA:
    :
    (%d)
```

Well, well. Looks like a network sniffer was running

```
:
(%d)
PKT: (%s %04X)
%s[%s] =>
%s[%s]
Log ended at => %s
%s: alarm
%s: getmsg
%s: alarm finished getmsg() = %i
c6Lqd3Dvn2l3s
(%s)UP?
filtering out smtp connections.
filtering out telnet connections.
filtering out rsh/rlogin connections.
filtering out ftp connections.
Usage: %s [-d x] [-s] [-f] [-l] [-t] [-i interface] [-o file]
-d int set new data limit (128 default)
-s      filter out smtp connections
-f      filter out ftp connections
-l      filter out rlogin/rsh connections
-t      filter out telnet connections
-o <file> output to <file>
Using logical device %s [%s]
Output to %s.%s%s
[Cannot bg with debug on]
Log started at => %s [pid %d]
rlogin
telnet
smtp
DATA LIMIT
TH_FIN
TH_RST
[edited for length]
```

Well, well. Looks like a network sniffer was running. The strings(1) output here matches up to a strings output for the old solsniff.c program that is floating around in cyberspace. This would indicate something really was going on that needs to be investigated.

This last conclusion required a leap of faith because I happen to have seen this output before. You may not be so lucky and may need to look at the binary in more detail, but from just looking at the strings output and the usage line in the suspect binary, I think anyone would agree that this binary was not supposed to be running. The fact that we proved it was not part of the vendor-shipped operating system using MD5 and the Fingerprint database would lead us to conclude, if nothing else, that we did have a problem and that the work we did rebuilding the disk from scratch was justified. From here we need to start looking for how the system was broken into in the first place.

At this point, I think we have exhausted the forensic lite scenario. To take things to the next level and delve into full forensics, we need to examine memory and swap to uncover evidence of how the intrusion unfolded. I'll leave that for a follow-up article on how dissecting swap may uncover exactly how the intrusion occurred.

loadable kernel modules

The New Frontier for Incident Response

by Keith J. Jones

Keith Jones is a computer forensic consultant for Foundstone. His primary area of concentration is incident response program development and computer forensics.

Keith.jones@foundstone.com

What would you do if traditional incident response tools completely failed during an investigation? That is exactly what I experienced when up against a loaded evil kernel module. Loadable kernel modules are changing the techniques used to perform an incident response because the level of compromise is raised from user space to kernel space. Once the compromise breaches the kernel space, the effects trickle down to any user-space executable resident on the trojaned system. This effect allows an intruder to change the behavior for any command executed on the system without changing the program binaries themselves. With this in mind, any trusted toolkit you transfer to the victim machine will also be automatically compromised. Therefore, I will explain how one malicious kernel module works and describe a couple of tools I developed to cope with the problem.

Overview of Loadable Kernel Modules

Loadable kernel modules (LKM) are a blessing for the system administrator, but a nightmare for an incident responder. LKMs were initially designed to provide dynamic functionality by altering a running kernel without rebooting. The slight altering of a running kernel can provide additional support for other devices such as new file system types and network adapters. Additionally, since kernel modules can access all functions and memory areas of a kernel, the depth of what it can alter reaches the whole operating system without any controls. Therefore, every function and memory resident struct is in danger of being compromised by a malicious kernel module.

One well-known malicious module for Linux kernels is named knark. Once knark is compiled and loaded on a victim machine, the syscall table is altered, which changes the operating system behavior. Basically, the syscall table is the entry point into the operating system provided to user-level programs and resides in kernel space. The formal definition of "syscalls" is given in manual section two of most UNIX operating systems. Whenever the kernel executes on behalf of user space, the area of an operating system a typical user executes in, the OS maps all of the commands and functionality executed on the command line to system calls within this table. Therefore, when knark alters the syscall table, it is altering user command execution. The important system calls knark alters are the following:

- getdents – This system call gets the directory entries (i.e., the files and directories) of a given directory. By compromising this call, knark is able to hide files and directories from user-level programs.
- kill – This system call sends a signal to a process, typically to kill it. This call is compromised such that an extra unused signal, #31, will trigger the option flags of a process to be set to the "hidden" state. When a process is hidden, its entry from the /proc file system is removed and therefore hidden from ps. Signal #32 unhides the file by resetting the option flags of the task.
- read – This system call reads the contents of a given file. Knark compromises this call such that it hides intruder connection specifics from netstat. The specifics are hidden because they are read from the /proc file system as files.
- ioctl – This system call changes the behavior of files and devices. When knark compromises this system call, it is able to clear the promiscuous flag on the network

With control of the OS, an intruder can make it return false information to the user-space queries

interface cards. Additionally, knark also inserts the functionality of hiding and unhiding files into this function.

- fork – This system call spawns a new process. When knark compromises this system call, it will hide all child processes created from a hidden parent process.
- clone – This system call spawns a new process. When knark compromises this system call, it will hide all child processes created from a hidden parent process.
- execve – This system call executes a file. It is called every time a command is entered at the prompt by a user. When this system call is trojaned, the kernel module can manipulate how and what commands are executed. knark allows an intruder to point one executable to another, similar to a symbolic link but without the evidence. When execve is compromised by knark, the destination executable runs instead of the expected source program.
- settimeofday – This system call sets the system time. knark compromises this system call by watching for predetermined clock-setting times. When one of these times is sent to this call to reset the clock, knark can either execute some administrative tasks or give the current user the user ID and group ID of root immediately. This eliminates the need of changing a shell to SUID-root in order to give root privileges to an ordinary user.

Since the syscall table has been compromised, the functionality of administrative tools is altered. netstat reports a network interface card that is never in promiscuous mode, and connections from given locations disappear. ps and top do not report hidden processes because they disappear from the /proc file system. ls ignores hidden files and directories. All of this occurs because when the tools are run they rely on the operating system to supply information. With control of the OS, an intruder can make it return false information to the user-space queries. This occurs without changing the binary files for netstat, ps, top, and ls. Therefore file system checksum tools, such as Tripwire, are useless against this type of compromise. Checksum tools are also defenseless against the executable redirection capabilities of knark. If an intruder were to link a file hackme to cat, every time cat is run the program hackme is executed instead. This allows cat to remain on the file system with the same MD5 checksum, yet execute with different functionality.

Furthermore, transferring a new set of tools to a victim machine with knark installed will not change the data reported. Even a trusted tool set must make system calls, therefore the tools become untrusted immediately when running on the victim machine. There is currently no way to circumvent a kernel-level compromise without using a toolkit that also enters the kernel space. This was my motivation to develop tools and techniques to check for the installation of LKMs and capture processes when a system may have a malicious LKM installed.

One caveat not previously mentioned is the existence of knark.o in the loaded module list reported by lsmod. Unfortunately for the investigator, there is a simple way to make this information disappear for the intruder. knark is packaged with another loadable kernel module named modhide, which makes itself and the last loaded module disappear. Once a module has disappeared, there is no way to unload it without rebooting the machine. Additionally, there is no easy method to even detect it is loaded, because all identifiable references to the module disappear. As has been shown, knark comes with all the tools to make it the ultimate stealthy rootkit.

Preventative Measures

If the ability to prevent a loadable kernel module compromise is available, it will obviously be the best solution. There are a few measures you can take to protect yourself from loadable kernel modules ahead of time. You can protect yourself from most of the maliciousness kernel modules can cause by securing your syscall table. A simple loadable kernel module can be constructed that watches the syscall table at periodic intervals and when other modules are loaded. If the sentry module discovers that the syscall table has been modified from its original state, it can alert the system administrator and even change it back to the original value. The following example code will work well with Linux 2.2 and 2.4. If you have a machine with more than one processor, it can be compiled by the following command: gcc –D__SMP__ -c syscall_sentry.c. If you have a machine with a single processor, just remove the –D__SMP__ definition. Once it is compiled, load it into the running kernel with insmod.

If the ability to prevent a loadable kernel module compromise is available, it will obviously be the best solution

```
/*
 * This LKM is designed to be a tripwire for the sys_call_table.
 */

#define MODULE_NAME  "syscall_sentry"

/* This definition is the time between periodic checks. */
#define TIMEOUT_SECS    10

#define MODULE
#define __KERNEL__

#include<linux/module.h>
#include<linux/config.h>
#include<linux/version.h>
#include<linux/kernel.h>
#include<linux/sys.h>
#include<linux/param.h>
#include<linux/sched.h>
#include<linux/timer.h>
#include<sys/syscall.h>

/* This function is a simple string comparison function */
static int mystrcmp( const char *str1, const char *str2)
{
      while(*str1 && *str2)
          if (*(str1++) != *(str2++))
              return -1;
      return 0;
}

/* This function builds a timer struct for versions of linux
 * less than Linux 2.4. It is used to set a timer
 */
#if LINUX_VERSION_CODE < KERNEL_VERSION(2,4,0)
/* Initializes a timer */
void init_timer(struct timer_list * timer)
{
            timer->next = NULL;
            timer->prev = NULL;
}
#endif
```

```
/* This is our timer */
static struct timer_list syscall_timer;

/* This is the system's syscall table */
extern void *sys_call_table[];

/* This is the saved, valid syscall table */
static void *orig_sys_call_table[ NR_syscalls ];

/* This function is needed to protect yourself */
static unsigned long (*orig_init_module) (const char *, struct module*);

/* This function checks the syscalls for changes
 * and changes them back to the original if it has
 * been changed.
 */

static int check_syscalls( void )
{
      int i;

      /* Add a new timer for our next check */
      del_timer( &syscall_timer );
      init_timer( &syscall_timer );
      syscall_timer.function = (void *)check_syscalls;
      syscall_timer.expires = jiffies + TIMEOUT_SECS * HZ;
      add_timer( &syscall_timer );

      for ( i = 0; i < NR_syscalls - 1; i++ )
      {
          if (orig_sys_call_table[i] != sys_call_table[i])
          {
              printk(KERN_INFO "\nSysCallSentry - sys_call_table has been
                        modified in entry %d!\n", i);
              sys_call_table[i] = orig_sys_call_table[i];
          }
      }

      return 1;
}

/* Check sys_call_table anytime a new module is loaded. */
static int long sys_init_module_wrapper( const char *name, struct
                        module *mod )
{
      int i;
      int res = (*orig_init_module)(name,mod);

      for ( i = 0; i < NR_syscalls - 1; i++ )
      {
          if (orig_sys_call_table[i] != sys_call_table[i])
          {
              printk( KERN_INFO "\nSysCallSentry - sys_call_table has been
                        modified in entry %d!\n", i);
              sys_call_table[i] = orig_sys_call_table[i];
          }
      }
      return res;
}
```

```
/* Module Init Code */
static int init_module (void)
{
     int i;
     printk(KERN_INFO "\nSysCallSentry Inserted\n");

     /* Initiate the periodic timer */
     init_timer( &syscall_timer );

     /* Save the old values of the sys_call_table */
     orig_init_module = sys_call_table[SYS_init_module];

     /* Wrap the init_module syscall. This will check to see
     * if any calls have been altered when a new module loads.
     */
     sys_call_table[SYS_init_module] = sys_init_module_wrapper;

     for ( i=0; i < NR_syscalls - 1; i++ )
     {
        orig_sys_call_table[i] = sys_call_table[i];
     }

     /* Start our first check */
     check_syscalls();
     return(0);
}

/* Module Cleanup Code */
static void cleanup_module (void)
{
     /* Return system status to the original */
     sys_call_table[SYS_init_module] = orig_init_module;
     printk(KERN_INFO "\nSysCallSentry Removed\n");
}
```

The current "state of the art" in LKM rootkitting is to modify the syscall table

The current "state of the art" in LKM rootkitting is to modify the syscall table. Therefore, this method of placing a sentry on the syscall table is practical because the syscall table changes so infrequently. Possibly the best true preventative measure you can take to protect your machines from this type of compromise is to remove the ability to load kernel modules completely. Production servers should have all the code they need to run compiled into the kernel, and loadable kernel modules should not be used.

There is another option available to protect yourself against hostile LKMs. A tool called "St. Jude," when compiled with one called "St. Michael," both guard against the modification of the syscall table, and checks root transitions for evidence of attacks, based on a ruleset created during a learning phase. (*http://www.sourceforge.net/projects/stjude*).

Development of Investigative Tools and Techniques

It is obvious that the investigation must examine the victim machine's kernel space in order to effectively respond to a kernel-level compromise. Therefore, investigators must change their tools and techniques. It will be assumed that the response to an incident involving knark will include a forensic duplication of the victim machine's storage devices. Therefore, any hidden files will be available to the investigator using that method. What the investigator will miss, however, are hidden processes and network information. This can be remedied by developing a kernel level "ps-like" tool that also retrieves executable images of each process. This tool will be a loadable kernel

Access to the process executable image in Linux is not trivial, but it is possible

module so that it can be loaded after an incident occurs. This section will describe, at a high level, one such tool and how it works to circumvent the problems involved with kernel-level investigations on a Linux 2.2 platform.

The most important struct for a kernel level ps tool is task_struct. It is a circularly linked list where every process on the system is present. Every action available for the process is available inside this struct, such as opened files, an executable image of the process, opened network sockets, file operators, and more. The following are several fields of important information for the investigator which will be written to a log file.

- *Process ID (PID)* – This is the unique number used to identify a running process.
- *User ID* – This is the user number that executed the process. It is important to know what privilege level the process is running as.
- *Process Status* – This flag indicates how the process is currently running. Since a process cannot occupy the processing power of the CPU all of the time, it may be sleeping. This flag will indicate what running state the process is in.
- *Process Name* – This is the human-recognizable name for the process. It is the equivalent to a portion of the command line used to execute this process.
- *Start Time* – This is reported in "jiffies," which is the number of system clock ticks since the machine was booted. This field is used to determine when the process was started. It is obvious when a process is initiated using a startup script during system boot because the number is relatively small. It may also provide more clues as to when the intrusion occurred.
- *Open File Handles* – Since everything in UNIX is a file, viewing the open file handles of a process allows you to see all open regular files, network sockets, and FIFOs. This information will be pertinent when tracking down processes that save information to files, like sniffers, or open network sockets, like back doors.
- *Command Line Arguments* – The command line arguments are available in the task struct and are useful when deciphering the options with which a process was executed. For example, imagine an intruder started netcat. It would be difficult to observe where it was connecting unless you had the command line arguments. The command line arguments available in the task struct would include the IP addresses and ports for netcat.
- *Process Environment Variables* – Each process that is executed has its own version of an environment table. Typically, it is a duplicate of the environment that the initiating user had at the time of the execution. Therefore, extra information of the intruder's session will be available by examining the environment variables available in the task struct.

Therefore, a tool would iterate through this circular linked list, saving the information for each process to a log file. This information would be very similar to the ps –ef command, so most investigators will be able to read it easily. Additionally, a separate file will be created for the process containing the executable image, also found in the task struct, for further offline tool analysis. Access to the process executable image in Linux is not trivial, but it is possible. The image resides in the memory map struct of the task. Within that memory map struct there is a virtual memory area associated with the task. Within that area, there is a virtual-memory file, which contains a file operator array. Once we have found the proper read function, reading the executable image and writing it out to another file is simple. Theoretically, a process should always have an executable image completely loaded in memory because it is possible to delete binaries from the file system after they are running. The most difficult part of acquiring the image is finding where it resides in memory.

While your module is running, no other process can be scheduled to run. Interrupts and other system activity can still take place, but the module has preempted execution. Because this module "freezes" the process list, I named it "Carbonite." The source code is available from *http://www.foundstone.com* and is a good basis to develop tools like it for later versions of Linux and different operating systems.

One last fact you can use to determine if knark is loaded on your system is to view the network card status. When knark is loaded, it never lets the network interface report it is in promiscuous mode. This is to prevent a system administrator from observing an intruder's sniffer, which places the card into promiscuous mode. What you can do, however, is simply run tcpdump or any other sniffer that uses promiscuous mode and view the status of the network adapter using ifconfig. If the card does not report that it is in promiscuous mode, knark could be loaded.

Conclusion

The ability to load kernel modules is a significant blow to incident responders. The malicious loadable kernel module is already publicly available and probably used in many intrusions. It raises the bar of compromise and incident response to the kernel level. Although it may seem that kernel modules are a significant factor when performing investigations, they are not lethal. You can see that there are simple preventative measures that you can take to protect yourself against this type of compromise. Furthermore, investigative tools and techniques for this type of compromise fight fire with fire by also executing in kernel space.

The ability to load kernel modules is a significant blow to incident responders.

implementing snort in a production environment

by Jon Lasser

Jon Lasser is a UNIX consultant in Baltimore, Maryland, and is the author of *Think UNIX*. He is lead coordinator for the Bastille Linux Project and thinks the four food groups are coffee, cheese, chocolate, and beer.

jon@lasser.org

Last year, I implemented Snort in a production environment at a medium-sized Web-hosting firm specializing in dedicated servers. We implemented Snort version 1.6 (later upgrading to version 1.7) on a system running Red-Hat Linux version 6.2. After we performed extensive tuning of the Intrusion Detection System (IDS) rule set and wrote a number of scripts to parse its output, the system proved quite useful for identifying denial-of-service (DoS) attacks. It was somewhat less useful for detecting actual intrusions, but even in this regard it was superior to running nothing at all.

Project goals were very vague, but the general idea was to increase the security and manageability of our network. Because we had no direct control over customer servers, we could identify misbehavior only by locating suspicious network traffic. A number of customers also ran IRC servers, making them prime targets for DoS attacks. Our hope was that by implementing an IDS we would be able to quickly identify the targets of DoS attacks. While we hoped to use the system to identify the attackers, addresses were almost always spoofed and the degree of cooperation with our upstream network feed did not lend itself to tracking down the attackers. (Many larger network providers will not even attempt to trace back spoofed packets unless you appear to be serious about prosecuting the attacker. They have limited resources to deploy, so this strategy makes sense from their perspective.)

Our system was a generic Intel-based Pentium III system running at approximately 700 megahertz with a 100-megabit Ethernet card. Since customers were permitted to perform virtually any activity on their dedicated servers, there was no site-wide firewall. The IDS sat on the VLAN used by the routers to talk to each other and sat on the "span" port so as to see all of the relevant traffic. However, the routers talked to each other over gigabit fiber, and traffic averaged approximately 120 megabits per second, so the IDS saw what amounted to a (hopefully) good sample of the traffic. Keeping up with the throughput given by the 100-megabit card kept the CPU at full utilization all of the time. Planned updates to the system included splitting the sensor from the processor so as to allow for expansion and to reduce CPU usage, but this was never implemented.

Linux was chosen as a platform not because of its strong packet-capture abilities but to remain consistent with the other servers on site. Simplicity of management was one goal of the implementation; there was no full-time security person on-site, so managing the IDS was one of a number of duties I had. Because the pcap library does not keep track of dropped packets on Linux, it is impossible to estimate with any accuracy the actual percentage of traffic processed by the server. It was certainly adequate for much of the task at hand. My rough guess is that the IDS saw three-fifths of the traffic. This guess is based primarily on what appear to have been complete sweeps of our network space: approximately three out of five target IP addresses in our range tended to show up in these sweeps. With even faster hardware now available inexpensively, it might be possible to capture all hundred megabits of traffic even without a more complex architecture; as some commercial vendors claim to be doing 200-300 megabits per second of actual traffic, it should be possible to push the systems harder. This may require more extensive engineering and more careful selection of rule sets.

Snort 1.6 and 1.7 built out-of-the-box on RedHat 6.2; newer versions of RedHat Linux have prebuilt Snort 1.7 packages available as part of their "PowerTools" collection of applications, though Snort 1.8 is now available. Snort was chosen due in part to my familiarity with parsing its output and the availability of other administrators in the area who understood and were willing to help with Snort. (Thanks, Andy!) Low cost of implementation was another consideration: the project goals were very vague, so justifying expensive software or hardware was out of the question.

The most sensitive part of the Snort configuration was the rule set used. A Snort rule set is a list of filters that apply to packets and determine which ones set off alarms. Rules can match packets based on source and target addresses and ports, particular protocols or flags, and the actual contents of the packet. The more rules implemented, the slower the system will be, and the more alarms will go off. It is difficult to strike a balance between a "quiet" IDS that misses important incidents and a "loud" IDS that finds such a quantity of suspicious traffic that actual attacks are lost in the noise. Our experience was one of trial-and-error, starting with an extensive rule set and whittling out those rules that became more trouble than they were worth.

A Snort rule set is a list of filters that apply to packets and determine which ones set off alarms. . . many DoS attacks will set off a large number of these rules simultaneously, lighting up the target system like a Christmas tree

We based our rule set on the one made available at *http://www.whitehats.com* with a number of substantial changes. First, because we were as worried about mischief originating from our network as that aimed at our network, we ignored the "internal" and "external" network definitions and modified all rules to consider any source and destination addresses as matching the given rule. We disabled all rules that matched solely on source and target ports; these rules have a very high false-positive rate. On the other hand, many DoS attacks will set off a large number of these rules simultaneously, lighting up the target system like a Christmas tree. Even with these changes, a large number of rules generated many false positives and were disabled. After that, we still had 10,000–20,000 individual alerts generated by the system on a typical day, many of which were false alarms that were too difficult to programmatically eliminate.

I wrote a group of scripts that ran hourly, examining the last full hour's incident logs. One script summarized all of the activity for the hour. It listed all of the rules that were triggered, sorted by the number of times each rule was triggered; it listed the top 10 source addresses of packets that triggered rules; and it listed the top 10 destination addresses of packets that triggered rules. This hourly summary gave me a quick read on the status of my systems over time and allowed me to find serious incidents not detected by the script described below. I also ran a daily summary script that provided the same information on a daily basis, allowing me to keep (essentially useless) statistics of how many detects a day we experienced.

Another script looked for particular alerts that we considered of high importance, such as those matching a particular exploit. The script then collated all attacks from that source IP over that hour – not simply the attacks serious enough to flag an email, but all detects from that source IP address – and mailed me the resulting log excerpt, with some boilerplate text up-front. These log excerpts often read like detailed summaries of scan-attack-scan scripts.

I used procmail to sort all of these messages into a single folder and set my mail reader to sort that folder by subject. Since each subject began with the IP address of the attacking system, it was easy to see attacks that spanned multiple hours. Using whois, it was quite simple to track down a responsible party for any particular server. By using Mutt's ability to act upon groups of messages simultaneously, and some clever vi

Frankly, whois information provided by most users is completely insufficient for accurate security-specific response

scripts to simplify stripping out extraneous information, I was able to quickly respond to a particular incident or group of incidents. However, the volume of alerts was such that I typically spent several hours a day sending "nastygrams" to sites that had scanned our network or attacked servers on it. The only solutions were to be more picky about what attacks and probes required a response or to automate further. Further automation was scheduled for the never-implemented second iteration of the IDS.

Frankly, whois information provided by most users is completely insufficient for accurate security-specific response. The one exception was Brazil's NIC, which included provisions for tagging an individual as the person responsible for security at a given site. I strongly urge the more widespread adoption of this technique, as it would both improve the usefulness of whois as relevant to security and to allow for further automation of security response. In general, the quality of information provided by the various registrars was abysmal: not a day went by without several security messages bounced due to invalid email information, and many more messages were misdirected due to changes in IP space management not reflected in the database. I also urge the use of a standardized "security" email address to simplify site contact for security-critical information.

The rate of response to security nastygrams varied wildly; some days as many as one-third of the sites to whom I sent complaints responded, while other days nobody at all did so. Responses were no more likely to come from places where English is the primary language than from other locations, and, in general, native English speakers were the least polite. However, they were also the most likely to request help in interpreting the logs or to offer to track down rogue server activity.

The most frightening activity picked up by the IDS was "low and slow" probes of our network: at any one time, there were probably five or six different sites involved in such probes. I have no way of knowing if these were coordinated attacks; because we never implemented a database back end for the IDS, I do not know if the regions scanned overlapped. The most obvious (!) of these attacks sent no more than two packets a day, inquiring of the version number of BIND running on a particular system. Even in the best case, it was hard obtaining sufficient evidence of malicious behavior to send to administrators at the appropriate sites: only over a four- or five-day period would enough packets accumulate that I would not feel foolish sending them to the administrator of the machine scanning our network. As I tended to go through the mail on a daily basis, I usually became aware of the low-and-slow probes when coming back from a long weekend. Had I tried to respond to all such attacks more frequently, I could have spent a full day every day doing nothing but processing logs.

As stated above, the most effective use of the IDS was tracking down the targets of DoS attacks. In fact, any packet-capturing system that can provide the source and destination IP addresses of individual packets can be used to do what we did. Simply, we had a script that started a Snort process which was configured to capture a particular number of packets (we used 10,000 packets in most cases) and print out the packet header information. We then extracted the source IP address of the packets, sorted them, and then used uniq to count the instances of any particular address. Finally, we did a reverse sort based on the packet count and printed the top 10 senders of packets.

Some high-volume customers continually ranked in the top 10, but there was usually an order-of-magnitude difference in packet counts when a DoS attack was underway.

Under normal circumstances, the top user might have 500 packets out of that 10,000; during an attack, the top user would likely have 3000–5000 packets. As a result of this, our "top 10" script was our most effective anti-DoS tool, followed closely by MRTG, which we could use to examine the traffic patterns of our top packet senders to determine whether or not the activity we were seeing was typical of that user.

As a means of detecting the actual compromise of servers, the IDS was rather ineffective. As most current exploits do not check for vulnerability before attempting to gain access to a system, we would frequently see exploits attempted against servers running different operating systems or even CPU architectures. Verifying the possible vulnerability of particular system for particular exploits was a Herculean task and, to be frank, boring. Worse, as we did not have root on most of our customers' systems, it was often impossible for us to verify whether or not a particular service was vulnerable on a given system.

This points to two areas that I believe need improvement in intrusion detection. First, I would like to see stateful systems that are not only able to detect a probable attack against a particular system but, following that, would be able to detect the success or failure of the exploit. This would require much more state than most intrusion detection systems maintain, and would likely be quite expensive in terms of memory and CPU usage. Nevertheless, it would allow much better prioritization of incidents, allowing limited resources to be deployed more effectively.

Second, it seems clear that intrusion detection is one small piece of the larger problem of incident response. This is the "what now?" problem: one of the rules has been triggered; what do you do now? This question goes unanswered by Snort and most other IDSes. After detection, the incident needs to be tracked, the system needs to be examined, and the offending system's administrator must be contacted. If the targeted system is not under your direct control, its administrator must be contacted, and so forth. I would like to see intrusion detection and response suites, essentially specialized trouble-ticketing systems, that would allow a group of detects to be classified as an incident and, once ticketed, allow that incident to be tracked. This would probably also include tracking the "owners" of blocks of IP addresses, including accurate contact information. This database could initially be populated via whois but could be refined by the system's users over time.

Overall, implementing a Snort-based IDS could not have been simpler. Were I to do it again, I would definitely use a separate sensor and a database back end: although the text-based logs were easy to process using Perl and shell scripts, the additional power of database queries might have turned up more useful information and saved a lot of time.

What was difficult about implementing an IDS with Snort was sorting the vast quantity of data produced by this system. Judicious selection of alert rules is crucial, as is some sort of automated post-incident processing. For my site, our hourly detect script was the most useful. It found all "serious" alerts and sent an email with all alerts, serious and otherwise, from that source IP. Even this was far overshadowed by the ability of the system to detect DoS attacks. That, however, could have been done on a much simpler system connected to the span port on the router; we simply did not think to use packet captures to detect DoS attacks until we had already implemented the IDS. (Before the IDS, we simply used MRTG to find spikes in network load on particular ports.)

This is the "what now?" problem: one of the rules has been triggered; what do you do now?

Unless a site has a specific need or a full-time security person on staff, I cannot yet recommend implementing an intrusion detection system. Although simple to implement, the data produced by such a system tends to be useful only to the extremely paranoid or those without adequate control of systems on the network in question. Over time, however, I expect that intrusion detection systems will become more powerful and more useful for the typical system administrator. Right now, however, these systems are too good at producing high volumes of data and not good enough at providing tools for turning that data into information.

pssst, wanna buy some network insurance?

by Peter Van Epp

Peter has spent the past 13+ years as a network/security/system administrator (in that order of priority) for Simon Fraser University in B.C., Canada.

vanepp@sfu.ca

From the title, you probably have the image of a seedy guy holding open a raincoat full of insurance policies (and an arm full of watches), but that isn't quite what I mean.

Think what you could do if you had a record of all network transactions that crossed your link to the Internet in an amount of disk space you could afford (without being a major government). There is an open software tool called argus that provides exactly that option. In this article, we are going to explore some of the things you can do with such information and state some reasons why you might want to have it, even if you can't interpret the results immediately. Hopefully, your network will become a safer place (and thus so will the Internet as a whole). You can certainly make friends with your internal/external auditors because this can be used to audit the effectiveness of a security policy, intrusion detection system (IDS), or firewall.

Let's start the ball rolling with a true war story (and the first good use of this tool). As the reader of our abuse email account in April a year or two ago, I received an external complaint about someone our way doing a port scan of a site. We are a university site, so nothing unusual there – just another contestant in the "you bet your account" contest. What was unusual was that a review of the weekend logs (this being Monday and the scan being Saturday) didn't turn up any scan. Then I looked at the date on the supplied firewall log: it wasn't from the previous Saturday but a Saturday last February. No problem: crank up the old disk machine, run the log files from last February through

ra (one of the argus data reporting tools included in the argus distribution) and, sure enough, there was a port scan. Thus I was able to tell the person reporting the "problem" that we had dealt with it the Monday after it happened (by declaring a new winner in the "you bet your account" contest) and that perhaps they should check their firewall logs a little more frequently than every two months or so.

Several points to make here: first, you should check the date and timestamp on a complaint using the argus logs to verify that a reported attack from your site really originated with your site (and wasn't forged as coming from your site), and that the reporting site (or you, when converting from a remote time zone) hasn't gotten the time wrong. This is important for things like dialup connections, where a different time zone can cause someone to be blamed erroneously and a slip up can be embarrassing and possibly expensive. Next, because you are recording all traffic, you can monitor outbound traffic from your site, which may or may not be important to you. Around our place, the standard comment on firewalls is that "the firewall faces inward to protect the Internet from our user community," not necessarily the other way around. On a commercial site, where you can (at least in theory) trust your user base, this may be less of an issue. However, in the case of a breach, the log provides you with a record of who else the attacker may have attacked from your site. Since the argus server can also be invisible to the network and tightly secured, chances are good the attacker won't find it and delete his tracks, even if he does so on the machines he broke in to.

A Hypothetical Case

While we are on this topic (i.e., sites that may have machines that are less than secure in large numbers) let's look at a hypothetical situation (which so far hasn't happened to me, and I hope never does). One morning a local law enforcement agent appears with a warrant to seize one or more of our machines on the grounds that they have been used in a DDoS attack. What are you going to do in this case? My guess is you're going to say, "Yes, sir, here it is, sir," because you can't refute the allegation (although even being able to do so may well not help until later in the face of a warrant).

I have a 50/50 chance. I can check the argus logs and either refute that we were the source of the DoS attack (best case) or confirm that we were the source (toast!). Because in the first case, while the reply packets are appearing on my Internet connection, there are no corresponding source packets issuing from my net, which means the source is being spoofed by a site without anti-spoof filters on their border router. If the attack did come from our site, our risk manager can choose to settle out of court or argue due diligence since we would have (and, in real life, have before it got this far) detected and dealt with the problem in a timely manner because of the argus logs.

This also illustrates another argus feature. When the analysis of the DDoS clients first came out in December a few years ago and indicated the control mechanism was an ICMP echo reply packet, the author provided a scan program that I was able to run on a segment monitored by argus. I determined what the scan looked like (an ECR flag on the flow rather than ECO indicating an echo reply with no corresponding echo request). Looking back, I noticed scans using ECR packets against my network starting in September of that year (luckily without success) and continuing even as I type this article and undoubtedly still going on as you read it. This procedure can be generalized to most new exploits: determine what the exploit looks like in the argus log and scan back over the time you (and more importantly your firewall and/or IDS) didn't know

I can check the argus logs and either refute that we were the source of the DoS attack (best case) or confirm that we were the source (toast!)

A firewall would ignore this, as they were outgoing HTTP accesses on port 80.

you should be looking for the signature to detect compromised machines. Knowing of a break in after the fact, while undesirable, is much better than not knowing of the break in at all.

A news break: as I type this (on July 19) the IIS Code Red worm has broken out. This provides a case study in using argus for unconventional things. A CERT notification yesterday gave me my first infected machine. A look at the argus log indicates a signature of many, mostly 0 length, connection attempts to offsite Web servers. A quick Perl script that takes the argus ra data reporting tool output as input and selects accesses to port 80 that are not on any of my local nets is quickly written. The source and destination IP addresses get stored in an associative array indexed by source IP address and then sorted. As long as the source address is the same, increment a counter (because this is a new destination access from this same source host). Once the source address changes, store the source address in a new associative array indexed by the remote host count. When the entire file has been processed, sort the array of counts in reverse numeric order and write it to standard out.

This is about a page of Perl and an hour or so of work. The output looks like this (with the addresses obscured to protect the guilty):

```
100539
        1xx.yy.zzc.65

271
        1xx.yy.zze.6

269
        1xx.yy.zzf.161

...
```

The first address (and 13 more like it across both our campuses) is a machine affected with the Code Red worm scanning other machines. The other two hosts are normal accesses. This is a fine example of why having a record of all the data passing through your network is very useful. A firewall would ignore this, as they were outgoing HTTP accesses on port 80. There are reports that the initial Snort rule, because of a rule mistake, wasn't catching all of these. Since argus is recording everything that comes by, the mark one eyeball (after some thinking and data processing) has no problem picking the difficulty out of the noise in the raw data. A manual human scan of the argus log verifies that this really is the worm from the ra output itself (again with addresses obscured):

```
Thu 07/19 06:56:44 s  tcp    1xx.yy.zzc.65.60806   ->   aaa.165.233.142.80   3  0  0  0  REQ
Thu 07/19 06:56:44 s  tcp    1xx.yy.zzc.65.60791   ->   bb.147.29.238.80     3  0  0  0  REQ
Thu 07/19 06:56:44 s  tcp    1xx.yy.zzc.65.60813   ->   cc.123.5.126.80      3  0  0  0  REQ
Thu 07/19 06:56:44 s  tcp    1xx.yy.zzc.65.60768   ->   dd.60.30.131.80      3  0  0  0  REQ
```

. . . (lots more just like this – 100,535 of them in fact . . .) So whack this host (and the 14 others) off the network.

Since we mentioned your firewall and IDS a couple of paragraphs ago (before I was so usefully interrupted), and since I promised at the start to make your auditor smile upon you, let's look at some of the ways argus can be useful in conjunction with these devices. While we all know none of us would ever misconfigure our firewall, in the hypothetical situation where this did happen, how do you demonstrate that your firewall/IDS is doing what your security policy specifies? One good way is to put an argus

server on both sides of it and perhaps use a tool such as tcpreplay to insert known attacks into the input data stream.

When you compare the two argus logs you would expect to see the attack packets in the outside argus log. If you also find such packets in the log from the argus server inside the perimeter, you have just found a problem. Of course, you also have logs back in time to see if the hole has bitten you already. In such a situation, I often find the standard operating procedure for swatch syslog watcher to be useful: filter out (one class at a time) the expected packets from the log output and carefully examine that which remains until you have explained why it should be there or have identified a problem to be dealt with.

Another nice feature of argus is that, since it is not deciding anything at all about the data stream but is just recording it in a compact form, it is harder – and with sufficiently large hardware, perhaps impossible – to overload the argus server. Thus, should the attacker be able to overwhelm your IDS or firewall, argus may still give you a record of everything that happened on your net. With argus you at least have the data; with only an overwhelmed IDS or firewall you don't (or at least not all of it). Something to think about, especially in terms of insurance.

With argus you at least have the data; with only an overwhelmed IDS or firewall you don't (or at least not all of it).

Now let's look in detail at some argus-detected incidents to get a clearer idea of some of the ways in which argus logs are useful. In all cases below, the addresses have been obscured to protect both the innocent and the guilty. The data is being read from files captured by the argus daemon program writing its output to a file (it can also output to a socket in real time if desired) and then later processed with the ra reporting tool. While argus and the reporting tools both accept tcpdump-style filters, the argus daemon is running with no filters in place (i.e., capturing everything). Note as well that these logs are from older versions of argus, so they will look a little different than current version logs.

This listing was generated using the argus ra reporting tool, which outputs a human-readable interpretation of the argus data. As mentioned, it accepts tcpdump-type filters; this listing was generated with a command line like this:

```
ra -r argus.log.file -c -n host bbb.cc.dd.75
```

which displays all records containing host bbb.cc.dd.75 with packet and byte counts (-c flag) and without DNS lookups (-n flag) on stdout. We see the root break-in, along with a large amount of additional data (including a port scan, which is what attracted my attention in the first place), extracted below. First, an explanation of what we are looking at. The first field is a timestamp, following that (blank in this instance) is a flag field which indicates things like retransmissions in a flow and a variety of other items documented in the ra man page. Following that is the protocol type (IP only in these old logs, many more in current argus versions). Then the source IP and port (the port being confusingly separated with a "."). Note that this is the machine that argus believes started the flow, either because it saw the SYN-ACK handshake, or it is its best (and sometimes incorrect) guess if it didn't see the original SYN ACKs. Next there is a flow indicator which, in this case, indicates a bi-directional flow (again the possible values and meanings are documented in the ra man page). Following that is the destination IP and port number. Then come the source and destination packet and byte counts (added by the -c command-line flag).

In version 1.8 (which this log is from) the counts are application data count; in the current version 2.02, by default, the counts are total packet length to allow traffic calculation, although the old meaning is possible with a command-line flag. The last field on the line is the connection-status field.

```
Fri 12/10 21:54:18  udp  aaa.255.11.140.2344  <->  bbb.cc.dd.75.111     1   1  64   36    ACC
Fri 12/10 21:54:18  udp  aaa.255.11.140.2344  <->  bbb.cc.dd.75.32774   1   1  200  40    ACC
Fri 12/10 21:54:07  tcp  aaa.255.11.140.3225   ->  bbb.cc.dd.75.23      12  10 62   123   CLO
...
```

Now knowing what we are looking at, we see that the attacking host aaa.255.11.140 made a connection from port 2344 to attacked host bbb.cc.dd.75 port 111, which happens to be the port mapper. The 36-byte reply from host bbb.cc.dd.75 told the attacker that rpc.statd was running on port 32774 of the attacked host. The attacker then sent the buffer overflow exploit code to port 32774 on the attacked host and created a new root account (from post-breach analysis of the system rather than strictly this log), although the successful telnet connection in the third line is also a good indication of a problem. The compromised machine was then used to port scan other machines on the Net, looking for more victims, which is what caught our attention.

Now, let's look at a DDoS attack and an echo/echo reply flow (as opposed to an echo reply only flow) for controlling DDoS clients. To create this data stream the following ra command line was issued:

```
ra -r argus.log.file -c -n net ccc.dd.245.0 mask 255.255.255.224
```

which selects only the hosts belonging to the 32-host network that comprises this address space and demonstrates some of the flexibility available in filters with argus.

The first line below is a normal echo flow from a network management station. Note the flow is bi-directional (<-> flag), and the status flag of ECO indicates a perfectly normal echo request/reply flow from ping.

```
Tue 05/15 16:09:55    icmp  aaa.bb.184.131  <->  ccc.dd.245.2  3 3      ECO
```

Sometime later, we see a unidirectional flow (-> flag) from control machine eee.ff.126.2 (somewhere in France) to compromised machine ccc.dd.245.2 (previously compromised via an unpatched RPC program) flagged as "only an echo reply" by the ECR end flag. Unfortunately, because this is an argus 1.8 log, we only get packet counts on the ICMP, not packet and byte counts as with argus 2.x. However, from both the acknowledgment ECR packet sent back to the attacking machine and the following DDoS and/or port scan, we see this is an attack. The first indication of a problem in this case was the abnormally large size of the argus log file due to the number of DDoS packet flows that were recorded.

```
Tue 05/15 16:16:20    icmp  eee.ff.126.2   ->  ccc.dd.245.2  4 0      ECR
Tue 05/15 16:18:39    icmp  ccc.dd.245.2   ->  eee.ff.126.2  4 0      ECR
```

Below, we see the DDoS attack; despite the varying source addresses in the same subnet (and many more being caught by the anti-spoof filters in the router before making it this far), all of these packets are in fact from machine ccc.dd.245.2. If there weren't anti-spoof filters on this subnet, then the source addresses would have been from all over the Net, making this a very hard attack to trace back. With anti-spoof filters, the attacked site would know to whom to complain, although it wouldn't point to the compromised machine. It would direct me to the appropriate subnet on my network because they are all appropriately anti-spoof filtered, and of course, the argus log would do the rest.

```
Tue 05/15 16:18:44   tcp    ccc.dd.245.25.2009 ?>  ggg.hh.141.177.71   1 0   0   0   TIM
Tue 05/15 16:18:44   tcp    ccc.dd.245.9.1012  ?>  ggg.hh.141.177.72   1 0   0   0   TIM
Tue 05/15 16:18:44   tcp    ccc.dd.245.30.1822 ?>  ggg.hh.141.177.73   1 0   0   0   TIM
Tue 05/15 16:18:44   tcp    ccc.dd.245.27.1204 ?>  ggg.hh.141.177.78   1 0   0   0   TIM
. . .
```

Now that we have seen some of the things argus is good for, let's look at what's needed to run it. It's likely a lot cheaper than you think, and it is certainly less than the traffic charges for your link. Up to a moderately fast link (such as FastEther or an OC3), a reasonably large PC, such as an 800MHz P3 class with 256 or 512MB of RAM and a 40 or 80GB UDMA66 IDE drive, will do fine. For Gig links you probably want something in the class of a Sun 450, and of course larger/faster is better (and even then, 200 to 400 megabits per second of data is as much as I am aware of being successfully captured on either argus or commercial IDS systems at present, although there may be faster systems out there). Benchmarking with iozone and bonnie disk benchmarks indicate that the 7200 RPM UDMA66 IDE drives can get as much throughput as a 36GB SCSI drive, so use what you've got and test! Argus runs under UNIX, so any of Solaris, Linux or the BSDs will do. Note on the BSDs, at least on current versions, that there is a bpf bug relating to the select system call, and you need to apply a kernel patch and rebuild the kernel (a source for patches is listed in references at the end of this article). There is a test procedure in the patch README file, and I recommend using it to check your setup so that you know whether you are seeing everything there is to see.

On the low end of the scale, a 33MB 386 (yes a 386) with 16MB of RAM and a SMC WD8013 10 baseT NIC running FreeBSD can keep up to 3 or 4 megabits per second of traffic, i.e., more than enough for a cable or a DSL connection at home. This was measured with tcpreplay, which we are going to discuss in a while. That also means that if you aren't blessed with a high-speed link (or money), any old machine (a surplus 486 for instance) may be able to keep up with your link, making experimentation or production cheap.

Although the capital costs are reasonable, there are additional costs. Interpreting the output data is a high-skill occupation at present, and I suspect it is likely to remain so. However, before I lose half the audience, let me point out that as an insurance solution this doesn't have to be a show stopper. As we have seen, the capital costs for argus are modest. Disk is cheap and getting cheaper. That means that even if you lack the expertise to interpret the output data, it is possible (and in my view very prudent) to collect, verify you are collecting (this is important and we will discuss how below), and save the data from your network. Should the worst happen and you have an incident, you can hire a consultant and have him or her interpret the argus data for you. Ideally, you will never use it, and it will sit on the disk unused until you decide it's old enough that you don't care anymore and overwrite it.

If you choose to do this for insurance, you will, on a regular basis, want to display the output of the argus log files with the argus ra data display tool to verify that you are in fact collecting data. The idea is to make sure something is in fact being collected from your network so that you don't discover when a disaster hits that the disk filled months ago and the argus data is lost.

In addition to the regular testing, when you first install argus and any time you increase the speed of your link or change the hardware argus is running on, you'll also want to run a test to make sure that argus can keep up with the maximum speed of

The open source tool called tcpreplay will replay a tcpdump file, either in real time (according to the timestamps in the file) or at a selected rate or (best of all) as fast as it possibly can given your hardware.

your link. This same thing (and the same tools) are useful for testing your firewall and IDS system. If you haven't stressed them, how do you know how they will perform when you need them to? More importantly, if they are going to fail, how do they fail: by passing all traffic (deadly if this is your firewall) or by ignoring attacks (deadly if this is your IDS system)?

The open source tool called tcpreplay will replay a tcpdump file, either in real time (according to the timestamps in the file) or at a selected rate or (best of all) as fast as it possibly can given your hardware. Tcpreplay has been added to the FreeBSD ports collection (at least in the 4.3-STABLE branch if not yet the release branch), which means that you can install it by just typing `make` in the appropriate directory under FreeBSD. In addition I have available both the original source (because the home site of tcpreplay is currently being reworked and a new version being produced) and a modification that will take two tcpdump files and two NICs and output a full duplex data stream. There are performance issues. I was only able to get about 160 megabits per second on a 600MHz P3 machine with a single UDMA66 IDE disk, but two disks on separate controllers may boost that. More importantly, there's a timing issue (which I currently punt): the two streams are not synchronized so it is possible that a response will appear before the corresponding source packet is sent on the other channel. The solution to this would be a tcpdump file compiler that would rearrange and/or insert packets to ensure that at full speed, data would come out in the appropriate order. This is very similar to a RISC C compiler having to insert `no ops` in the instruction stream to maintain memory access timing. In this case, you need to either move a later packet or insert a padding packet to ensure that, when played at full speed, the response will appear after the request in the other data stream. It isn't rocket science, but it is work that I currently haven't had time to do. Again, if someone is interested in writing it before me I'd be happy to test.

This setup is what identified the bug in the `bpf` routines of the BSDs with the select system call during a test of argus. It's also what I used to determine that Solaris and (of all things) Linux are able to capture data at a full 100MB (Solaris cheated by using an E450 rather than a PC; I haven't run the test on Solaris 86 yet). Surprisingly, the BSDs (at least FreeBSD 4.2 and OpenBSD 2.7) lose a small percentage of the traffic at 100 (on the identical hardware that Linux was using). It is also the setup that verified that a 386 can keep up with 3 to 4MB of data without losing anything. Being able to repeat an exact sequence of packets (and to vary the speed of delivery) is a very powerful test tool, not only for argus but also for your firewall, IDS, sniffers, and network gear.

Deploying Argus

Now that you have implemented argus and verified its performance, let's look at some of the issues with deploying it. First, it is a juicy target. It contains lots of sensitive data about your network all in one convenient place, so you need to secure it heavily. It is also potentially a privacy issue, and you need to have appropriate approvals and policy around what will and won't be done with the data. Since argus by default is only looking at the headers, the issues are somewhat less problematic than with an IDS that is also looking at the data, but in either case the issues still need to be addressed before implementation proceeds.

Like your IDS system, it only needs read access to the network. That suggests that you should use a splitter such as the Shomiti Century tap (for 10/100 baseT) or an 80%/20% optical splitter for a fiber connection to isolate your systems (both argus and

an IDS). You should tighten down the UNIX box it runs on to run either nothing at all (which implies management access from the machine console and is likely too paranoid a reaction in many cases) or with only a well-patched SSHD running on a private network deep in the well-protected heart of your network. This is likely the configuration to run. It allows you to move the captured data (via SCP for instance) to another heavily secured (and large-memory) machine where you want to run the analysis programs. This prevents the analysis process from stealing resources from the capture process, which in turn may result in packet loss during capture. If your link speed is not that high and you have a fairly large machine, this may not be an issue, but it is something to keep in mind in any case. Remember to secure the backup tapes (assuming you are archiving to tape) as well; the data on them is just as sensitive as the data on the disk (and more often overlooked when security is being considered).

If you are on a busy/fast network, you probably need to increase the bpf buffer size in your kernel (the default is usually 8,000 or so with a max of 32,000, at least on the BSDs). Boost it to the full 32,000 and larger, if possible, to give yourself the best chance you can of capturing all the packets. This is where the "packets lost by kernel" message in argus (and other libpcap-using programs) comes from. Note that even if this number is 0 it is still possible that your network interface is silently dropping packets at a point before this counter in the bpf routines, so you need to use a tool such as tcpreplay to send a known data stream at the maximum rate to determine that your installation can keep up with the maximum data stream.

Once you have done all of these interesting (and time-consuming) things, and argus is happily collecting data, what kinds of things do you want to look for (and, better yet, write tools to look for automatically) in the data?

The prime one comes under the label of "history." As you get more and more logs from your network, you can also get a baseline of what is normal for your network. New patterns of access (in or out) are of interest. If there is suddenly a connection to a machine that hasn't occurred before, either someone new has been granted access to the machine or there is a problem. Politely asking the owner of the machine (or alerting the owner to the change automatically) can catch a problem early.

A connection in from the outside to a machine followed by a connection to a machine on the outside is a common cracker trick for laundering connections to avoid detection. Such a pattern (which again a firewall isn't likely to flag since it looks legit) deserves to be asked about to make sure it is authorized. Again, more of concern on a university campus, port scans both from the outside world heading in and from the inside against external hosts are of interest (and usually automatic first prize in the "you bet your account" contest if done from my site). With experience, you begin to be able to pick out the tool being used to scan you from the pattern of data in the logs.

One interesting feature of an argus log is the ability to detect slow port scans. I've seen some where that one probe is done around every 40 minutes for weeks. I say they are of limited interest because the event of interest isn't the scan, but any machine that is compromised, which will typically show up as a much more immediate log entry.

One use of scan (including slow scan) detectors is to populate a "suspicious IPs" file. This consists of IP addresses that have scanned or attempted compromises against your network. Future connections from such a site are "interesting"; they may also be suspicious, but they are certainly worth paying attention to because they may be coming back to exploit a hole that they have found on an earlier scan, or they may be a dif-

As you get more and more logs from your network, you can also get a baseline of what is normal for your network.

Since argus captures the number of bytes transferred in both directions and the TCP flags, it is possible to see (and thus to automatically filter out) connections that failed, reducing the amount you need to consider suspicious.

ferent customer of an ISP or cable company (with DHCP) doing something perfectly legit such as accessing a public Web page. As long as you remember that and don't overreact, it's fine to be suspicious of such an access, and, sometimes it will net you a cracker, a compromised machine, or both.

Since argus captures the number of bytes transferred in both directions and the TCP flags, it is possible to see (and thus to automatically filter out) connections that failed, reducing the amount you need to consider suspicious. Again, because argus records the byte and packet counts of all connections by post-processing them, you can acquire a complete traffic record (down to the source / destination IP / port level, if desired) in your network. This is interesting, as mentioned above, because of patterns and history. If you process traffic counts by machine and sort them (usually reverse order is the most interesting) and keep history of previous "normal" (for some value of normal) traffic patterns, the result of a compromise will leap out at you (along, unfortunately, with a number of false positives). If a machine that has previously done almost no traffic off-site suddenly is transferring large amounts of data to machines all over the net, you can be reasonably sure you have a compromised machine running a warez site or someone (perhaps the authorized user) running one of the distributed file-sharing programs.

I find that a query along the lines of "What has this to do with university business?" or "What account number should the bandwidth charges be charged to?" will either cause an abrupt halt or provide a valid explaination (more usually the former than the latter). Basically, what I'm saying here is that for external compromises (of any kind) to be useful, the attacker has to change the traffic profile of the machine (otherwise, it is of no use to them). When they do that and you have a complete log of the traffic, it becomes detectable and stoppable with limited damage. There isn't any easy way around this, other than, of course, finding a less protected system somewhere else.

This is another great advantage of a complete connection history: you can step back in time and see if you have been compromised by a newly discovered (on the white-hat side of the fence) exploit before you knew it was an exploit. Again, while it would be better to find this before the compromise (and it may show up in the changed traffic pattern before the compromise itself surfaces), better late than never (or when the lawyer and/or cops are pounding on the door). Your complete log can also absolve you of attacks with forged source addresses (your forged source address!) since the reply packets from the attacked host will show up in the argus log, but there will be no corresponding attack packets in the log from your site.

There are a couple more classes of traffic, such as a large amount of apparently purposeless traffic to or from a site. Unfortunately, on a university campus this is often hard to differentiate from legit research traffic which often seems to have no purpose. I remember what turned out to be a DoS attack directed at us coming from a university machine with ldap in its hostname on port 500 (X.500), which could plausibly be someone doing an X.500 update across the net as research. This may be a DoS or DDoS attack. We have seen a number of these involving the DNS, queries with (I assume) a forged source address for a largish DNS record repeated over and over.

I've thought about (but not implemented) an authenticated Web page that could be accessed by the sysadmin of a machine or network which would display the history data of connections and possibly traffic patterns to and from their machines. There are privacy issues here that would need to be carefully considered and debated within the

user community (and it may well need to be opt in), but it may make a very powerful tool. If this interests you, feel free to join the argus developers list and implement.

Finally, what challenges does the future hold? The main one is the increase in link speed (without a corresponding increase in memory, disk, and CPU speed). That will make it much more difficult in the future to be able to keep up with the higher speed links and will make life very, very challenging. One solution is to divide the link traffic up among IDS/firewall/argus boxes using a demultiplexor. The problem is that a skillful attacker who is aware of that can arrange to flood enough traffic at your box to possibly slip an attack by you. It is possible to keep up with a 100baseT/OC3 Internet link (that's what I have now). When my link speed goes to gigE, and then possibly 10GigE (since the underlying backbone is going to OC192), how do we monitor it (or more correctly, who has that somewhat used Cray?). We most certainly will be living in interesting times.

REFERENCES

Argus: *http://www.qosient.com/argus*

tcpreplay: *http://www.anzen.com/research/nidsbench* (not available at the moment so also at *ftp.sfu.ca /pub/unix/tcpreplay)*

bpf patches: *ftp.sfu.ca /pub/unix/argus*

Century tap: *http://www.shomiti.com/*

optical splitters: *http://www.netoptics.com/*

survivability with a twist

by Sven Dietrich

Sven Dietrich is a member of the technical staff at the Carnegie Mellon Software Engineering Institute in Pittsburgh, PA. When he does not snort packets, he conducts research in computer security and survivable network technology.

spock@cert.org

Glancing at the hexadecimal pattern on the screen, I was quite happy with my intrusion detection system: I had netted some Ramen worm exploit code and handed it over to the appropriate incident response team. It had caught itself in one of the flytraps that was listening on the border network. This was quite surprising in light of the stumbling blocks I had encountered in the preceding few weeks.

The Mission

This catch should not have been that surprising. During the 18 months preceding this incident I had built up a network analysis and intrusion detection system (IDS) geared toward dealing with distributed denial of service (DDoS) attacks.[1] One of the difficulties I had encountered during this time was maintaining the set of systems to the point of self-sufficiency and self-repair, within reason at least, under common attack scenarios.

One of the fears of a security expert is the loss or absence of network forensics. "Network traces? Sorry, the network analysis systems are down/not available/not up yet." Network forensics provide clues to a security expert in the same way a detective would use clues to solve a mystery or a crime. Typical examples are packet flows collected at the router level, TCP-wrapper[2] logs at the host level, and intrusion detection system logs. Frequently, an intruder will try to disable monitoring systems prior to launching the real attack so that it cannot be reconstructed. The mission is to be able to reconstruct the incident, if not fully, at least partially.

Departure from Basic Security Concepts

The computer security field assumes a binary model, where a system either resists the attack or is compromised. Many years of research have gone into the effort of building higher and higher walls that the intruders cannot overcome. This inevitably leads to an arms race with the attackers, as they attempt to undermine, breach, or otherwise transgress those walls.

Rather than try to compete in this game, the field of survivability takes a different spin. The concept of survivability, as currently defined, is the ability of a system to fulfill its mission in the presence of attacks, failures, and accidents.[3] In the survivability worldview, one accepts the compromise or failure of one or more components of the system, as long as the mission can be completed, even if a reduced or degraded mode is necessary. The field explores various techniques for building survivable systems, including techniques borrowed from dependability and fault-tolerance, among others. For more philosophical foundations on the definition of the term, please refer to the references.[4]

Motivation

As I had investigated some initial incidents, I was often faced with the frequent absence of evidence. I wanted to be able to analyze most aspects of the attack, whether it was denial of service or not. Therefore, I needed a system that could operate, even partially, in the presence of a DDoS attack to gather as much information on the attack as possible. Even the local legitimate "security scans" that would periodically check for unpatched and insecure systems could rattle the hosts quite a bit.

The Approach

Since my budget was small, my choices were somewhat limited. I decided to take baby steps and proceed empirically. Based on my first setup (a sensor piping data back to a

data collection host over a secure shell connection), I identified operating systems and hosts stable enough for my purpose. However, I could not stop there. That summer was very hot and the local power company kept implementing rolling brownouts and blackouts. I wanted to look at all the components of this little system: what hosts would come back up and how would they recover after the "uninterruptible" power system lost power?

How robust was my setup really? I had found several uninterruptible power systems and distributed them among the hosts. As the system lived in a set of racks, colleagues would repeatedly trip over power and network cables as they were trying to access their own prototype systems. Of course this kept me on my toes. I had managed to build a set of redundant systems – replication of services, excellent! A geographic dispersion of the systems and their associated data files followed to minimize the impact of a local failure. My next step was to create a heterogeneous environment, so that one potential attack would not necessarily affect all of the systems involved, which was achieved by selecting a slightly different operating system for each host. (For those die-hard operating system aficionados who must know, the two operating systems were OpenBSD and NetBSD.)

Slicing the Network Stream

Since a complete loss of disks, systems, or other components could not be excluded from consideration, I wanted to record the data in several ways, so as to "rebuild" events later. It was absolutely critical that I be able to return to any given point in the data sets so I could perform the aforementioned network forensics. Gathering packet headers using argus,[5] full-packet recording using tcpdump,[6] keyword detection across protocols and ports using ngrep,[7] and signature detection and anomaly detection using Snort[8] were some methods for the slicing. Effectively, one performs several types of data reduction while one drinks from the fire hose that is the network stream.

Some types of attack, such as a buffer overflow in tcpdump, could disable, compromise, or blind intrusion detection systems. These so-called in-band attacks would travel in the data stream that one is recording and would act on a listening tool itself or affect a susceptible operating system, causing it to hang or crash. By varying the way one looked at the data, the chances that at least a few monitoring tools would remain orthogonal to the problem were relatively high. Even that was not sufficient; I wanted to reconfigure my system in the event of loss, corruption, or compromise of one of my components.

The Interconnection Network

In order for the different hosts to exchange status information, network stream data and logs, I built a redundant interconnection network, opaque to the outside observer. By performing queries, from the simple pinging of a host to a more complex one such as interrogating via a series of TCP/IP client-server messages whether a particular service was still running, the other hosts were given a view of the system as a whole. In the event of a non-response, the remainder of the system could then reconfigure by firing up appropriate replacement processes, such as a new packet logger or signature detector, without impacting the current role of the host. Each host had a primary role assigned to it, with a list of secondary and tertiary roles that would be assumed in case of the loss of a primary-role host. Several strategies were conceivable. One very simplistic one, noticing that a primary-role host had vanished and taking the role of that host if the local host was capable of doing so, seemed to work well enough. Another

These so-called in-band attacks would travel in the data stream that one is recording and would act on a listening tool itself . . .

REFERENCES

1. Sven Dietrich, "Scalpel, Gauze, and Decompilers: Dissecting Denial of Service (DDoS)," *;login:* (November 2000), theme issue on security.

2. Wietse Venema, "TCP Wrapper – Network Monitoring, Access Control, and Booby Traps." 3rd USENIX Security Symposium, Baltimore, MD (September 1992), *http://www.porcupine.org/*.

3. R.J. Ellison, David Fisher, Rick Linger, Howard Lipson, Tom Longstaff, Nancy Mead, "Survivable Network Systems: An Emerging Discipline," Software Engineering Institute Technical Report No. CMU/SEI-97-TR-013 (November 1997).

4. CERT Survivability Research page, *http://www.cert.org/research/*; John C. Knight, Matthew C. Elder, "Fault Tolerant Distributed Information Systems," International Symposium on Software Reliability Engineering, Hong Kong (November 2001); Jonathan Millen, "Local Reconfiguration Policies," IEEE Symposium on Security and Privacy, Oakland, CA (May 1999).

5. argus: *ftp://ftp.andrew.cmu.edu/pub/argus/*.

6. tcpdump: *http://www.tcpdump.org/*.

7. ngrep: *http://sourceforge.net/projects/ngrep/*.

8. Marty Roesch, "Snort – Lightweight Intrusion Detection for Networks," USENIX LISA XIII (December 1999), *http://www.snort.org/*.

9. Sven Dietrich, "AMPLIFIDS – A Survivable Ensemble," in preparation.

one, pushing out the newly learned information to the remaining known hosts, led to a collaborative decision.

Network time information, a source of accurate time which is important both for correlating events across the globe and coordinating between hosts, relied on two separate sources: a radio clock receiving the official time signal and a GPS-based clock, both on local networks. These timings were crucial for the analysis of the collected exploit and attack data. More details describing this system can be found in an upcoming paper.[9]

The Twist

Sitting in deep thought in front of my monitor, I recalled my first day back after my vacation. Two of my components, computationally powerful yet non-critical as they were, had for unexplained reasons been disconnected from both the interconnection and the border networks. I had discounted it as a mistake I had made since I had put the system into survivable mode before going on my vacation. Actually, it turned out to be a confirmation of the security credo that insider threats account for a significant number of the problems. As the non-critical components were removed from the system, the monitoring system proceeded to reconfigure itself in order to compensate for that loss and make the mission survive.

What I had designed to work against the outsider threat ended up working against an insider threat, intentional or not, and it was put to the test in a perfect setting: I had gathered sufficient network data to understand the Ramen incident and suggest mitigation measures.

The mission had succeeded.

a secure OS-based firewall

System administrators working with limited resources must be resourceful. Sometimes, however, this same limitation can force the system administrator to think thoroughly about the problem at hand in order to utilize the scarce resources effectively. In this article, I present an example of how a limitation in disk space led me to rethink the role of the firewall. I also show how to build a BSD-based firewall that fits on a floppy and runs on a PC without a hard disk. This type of configuration could be, in my opinion, more secure than some commercial products and most OS-based firewalls running on a PC with a hard disk drive.

by Oscar Bonilla

Oscar Bonilla is an instructor at Galileo University in Guatemala. As part of his consulting career, he has designed the networks of three major ISPs in Guatemala and El Salvador. His main interests include operating systems, computer networks, and Internet security.

obonilla@galileo.edu

For the past few years, I have been working as an instructor for the computer science department of Galileo University in Guatemala (*http://www.galileo.edu*). As is often the case with universities in undeveloped countries, the teaching staff has to take over other responsibilities usually given to administrative staff in universities with more resources. Since the subjects I teach are operating systems and computer networks, I have been implicitly expected to be the system administrator for the computing infrastructure of the university. Universities in third-world countries usually have very limited resources for computer infrastructure. For example, the primary firewall of the university I work at, a PC running FreeBSD and IP Filter, crashed a couple of months ago and needed to be replaced. I found a machine lying around that had most of the specifications needed for the job, except for a hard disk drive. I asked the supplies department for a 40GB hard disk drive and got a 1.44MB floppy.

I asked the supplies department for a 40GB hard disk drive and got a 1.44MB floppy.

This lack of resources forces system administrators to think carefully about how the available resources will be used. In the firewall example above, having limited space makes you to think very carefully about what programs you will be installing on the machine and the uses you will give them.

A firewall is essentially a discriminating router. It receives IP packets on one network interface, tries to match the packet with one of the rules, and takes an action which could be routing or dropping the packet. All of this is done inside the kernel. Thus, firewalls do not really need many programs to accomplish their job. All that is needed is the kernel, a program for installing the firewall rules, and a few commands for initializing the network interfaces, turning routing on, and checking how the firewall is doing.

Most operating systems come with many programs and services installed; many of these services are even enabled by default. This is not surprising since they are general purpose operating systems and have to be useful to a wide variety of users. In the next section we will see how we can choose a minimal subset of these programs that lets a firewall do its job.

The One-Floppy Firewall (using PicoBSD)

PicoBSD is a variant of FreeBSD that fits in a single floppy. It lets you create a boot floppy that contains a custom kernel and a memory file system in which you can install your own subset of the programs available in the FreeBSD distribution. You can also add your own programs as long as there is still space in the file system.

You must have a FreeBSD system with full sources in order to build a PicoBSD floppy. This is because PicoBSD utilizes a technology called "crunched binaries." This means that several programs (and libraries) are combined in a single statically linked binary, thus saving considerable amount of disk space. This statically linked binary uses its

The bigger you make the MFS, the bigger the kernel will be, and the less memory you'll have for running user processes.

own name (argv[0]) to know which function it must perform. You only need to hard link it to every program name you have "crunched" in it to have it execute the proper functions. In order to build the crunched binary, all the binary files must be recompiled using special flags. The reason we hard link the different program names to the crunched binary, as opposed to soft linking it, is that a hard link only uses an entry in the directory for storing another name for the same inode, whereas a soft link uses another inode with the path of the linked program. Since inodes take space in the file system, it is more efficient to use hard links.

In FreeBSD, all of the required sources for PicoBSD are in the directory /usr/src/release/picobsd. In this directory you'll find several examples of various one-floppy configurations that do things such as routing, dial-up serving, etc. However, we'll see how to build a custom PicoBSD floppy with only the programs you want.

A typical configuration directory has the following hierarchy:

```
floppy_name/
        PICOBSD
        config crunch.conf
        floppy.tree/
    etc/
        ...files that will go in /etc on the floppy...
    root/
        ...files that will go in /root on the floppy...
mfs_tree/
    etc/
        ...files that will go in /etc on the MFS...
```

In the picobsd directory there are several examples that you can copy and modify to suit your needs. Alternatively, you can create the configuration directory from scratch. I will explain here how to create the hierarchy from scratch.

The floppy_name is the name of the directory that will hold all the configuration files for the floppy. Think of it as a project name. I chose to call it offw (One-Floppy FireWall).

The PICOBSD file is a FreeBSD kernel configuration file specifying which device drivers and options to install in the new kernel. It must start with the following lines:

```
# Line starting with #PicoBSD contains PicoBSD build parameters
#marker        def_sz   init     MFS_inodes      floppy_inodes
#PicoBSD       3500     init     4096            32768
options MD_ROOT_SIZE=3500     # same as def_sz
```

The first two lines are just comments. The third line is a comment for the kernel configuration program config(8). However, that third line tells the PicoBSD building script the parameters for building both the MFS and floppy file systems. The four parameters def_sz, init, MFS_inodes, and floppy_inodes specify the size of the MFS file system, what program to use as init, the number of inodes to use in the MFS, and the number of inodes to use in the floppy file system, respectively.

PicoBSD uses two file systems for operation: MFS and a floppy file system. MFS is a Memory File System that's patched into the kernel and loaded at boot time to the machine's RAM. Since MFS is patched into the kernel, it makes the kernel binary image bigger. The bigger you make the MFS, the bigger the kernel will be, and the less memory you'll have for running user processes.

The other file system used by PicoBSD is the floppy file system. This is the file system that remains on the floppy. In this file system, you'll have the kernel itself and some files in which modifications must persist between reboots. Another program that will be copied to the floppy file system is the kernel binary image itself. What this means is that as the kernel image gets bigger (because the MFS is bigger or because you have too many drivers in the kernel), the space available in the floppy file system will get smaller.

You have to achieve a balance between the size of the MFS, the drivers configured in the kernel, and the files you put in the floppy file system. The bigger the MFS, or the more drivers you have in your kernel, the bigger the kernel binary image that will be copied to the floppy file system, thus leaving less space for programs and files in the floppy file system.

So how do you go about choosing the right sizes? I basically put as much as I can on the MFS and leave the floppy file system only for configurations files and data that must persist between reboots. There are two reasons for doing this: (1) if you put files in the floppy that will be used a lot (like binaries), the floppy must be inserted and working in order to use those files, and (2) floppy access times are usually orders of magnitude slower than memory access times. So in the end, I've found that it works best to keep most of the files in the MFS and leave as little as possible in the floppy file system. Files that need to be modified easily (without recompiling the kernel image) should go in the floppy file system. For example, the firewall rules configuration file is a good candidate for the floppy file system.

Getting back to the PICOBSD kernel configuration file, the rest of the configuration lines will be what drivers you need and what options you want set in your kernel. You can find plenty of information on configuring FreeBSD kernels on the FreeBSD Handbook, available from the FreeBSD home page (*http://www.freebsd.org/*). Just remember to keep things to a minimum. Here's my complete kernel configuration file:

```
# Line starting with #PicoBSD contains PicoBSD build parameters
#marker         def_sz  init    MFS_inodes      floppy_inodes
#PicoBSD        3500    init    4096            32768
options MD_ROOT_SIZE=3500      # same as def_sz
# the machine architecture
Machine         i386
# the cpu type
cpu             I686_CPU
# an identifier for this kernel
ident           FIREWALL
# maxusers sets the static sizes of various structures inside the
# kernel, like maximum number of open files, etc.
maxusers        12

options         INET            #InterNETworking
options         FFS             #Berkeley Fast File System
options         FFS_ROOT        #FFS usable as root device [keep this!]
options         MFS             #Memory File System
options         MD_ROOT         #MFS as root
options         COMPAT_43       #Compatible with BSD 4.3 [KEEP THIS!]
options         PCI_QUIET

device          isa0            # ISA Bus
device          pci0            # PCI Bus
```

. . . it works best to keep most of the files in the MFS and leave as little as possible in the floppy file system.

```
# Floppy disk controller
device          fdc0    at      isa?    Port IO_FD1     irq 6     drq 2
# Floppy disk
device          fd0     at      fdc0    drive 0

# Keyboard controller
device          atkbdc0         at      isa?    port IO_KBD
# Keyboard
device          atkbd0          at      atkbdc? irq 1
# VGA display
device          vga0    at      isa?
# Console Driver
device          sc0     at      isa?
# Math Coprocessor [KEEP THIS!]
device          npx0    at      nexus?  port IO_NPX     irq 13
# Serial Port
device          sio0    at      isa?    port IO_COM1    flags 0x10   irq 4
# miibus is needed for certain Ethernet cards (like 3Com FastEthernet)
device          miibus
device          xl0             # 3Com FastEthernet Card
pseudo-device   loop            # local loop interface (lo0)
pseudo-device   ether           # Generic Ethernet Drivers
pseudo-device   pty     8       # Pseudo TTY's
pseudo-device   md              # memory disk
# Berkeley Packet Filter (not needed if you don't need tcpdump)
pseudo-device   bpf 4           # 4KB, for tcpdump
# IPFilter is the packet filter we'll use
options         IPFILTER
# Logging facility for IPFilter
options         IPFILTER_LOG
# Make IPFilter block all packets by default
options         IPFILTER_DEFAULT_BLOCK
options         PROCFS          #Process file system
```

The next file you need to create is config. This file is a configuration file that is sourced by the PicoBSD build script. It must contain only variable definitions. The one important variable that must be in this file is MY_DEVS, which tells the build script which devices to create in /dev inside the MFS. This is done passing each item in MY_DEVS to the standard FreeBSD MAKEDEV script usually found in /dev. This is what I have:

```
MY_DEVS="std vty10 fd0 pty0 cuaa0 bpf0 bpf1 ipl"
```

This example tells MAKEDEV to create standard devices (std), 10 tty's (vty10), a floppy disk drive device (fd0), the pseudo stty's (pty0), a serial port device (cuaa0), two Berkeley Packet Filter devices (bpf0 and bpf1), and an IPFilter logging device (ipl).

The ipl device is particularly important because it's the interface between IPFilter inside the kernel and the command line utilities that run in user space and are needed to load the firewall rules. The bpf devices can be left out if you don't need tcpdump in the firewall host.

The next file is called crunch.conf and contains the specifications needed to make the crunched binary.

The type of directives supported are (from the crunchgen(1) man page):

- srcdirs dirname: A list of source trees in which the source directories of the component programs can be found. These dirs are searched using the BSD <source-dir>/<progname>/ convention. Multiple srcdirs lines can be specified. The directories are searched in the order they are given.
- progs progname: A list of programs that make up the crunched binary. Multiple progs lines can be specified.
- libs libspec: A list of library specifications to be included in the crunched binary link. Multiple libs lines can be specified.
- buildopts : A list of build options to be added to every make target.
- ln progname linkname: Causes the crunched binary to invoke progname whenever linkname appears in argv[0]. This allows programs that change their behavior when run under different names to operate correctly.

Here's what I have in my crunch.conf file:

```
# We don't need PAM, NETGRAPH, IPSEC or INET6 (and we'll hint the
# sources that this is a RELEASE_CRUNCH
buildopts -DNOPAM -DRELEASE_CRUNCH -DNONETGRAPH -DNOIPSEC -\
        DNOINET6
# directories where to look for sources of various binaries
srcdirs /usr/src/bin
srcdirs /usr/src/sbin/i386
srcdirs /usr/src/sbin
srcdirs /usr/src/usr.bin
srcdirs /usr/src/gnu/usr.bin
srcdirs /usr/src/usr.sbin
srcdirs /usr/src/libexec
# Some programs are especially written for PicoBSD and reside here.
srcdirs /usr/src/release/picobsd/tinyware

# init is almost always necessary.
progs init # 4KB.
# Without ifconfig you wouldn't be able to configure IPs on your
# network interfaces.
progs ifconfig  # 4KB.
 # You need a shell.
progs sh      # 36KB.
ln sh -sh
# These are just some utilities I find useful.
progs echo      # 0KB.
progs pwd
progs mkdir rmdir
progs chmod chown
progs mv ln # 0KB.
progs mount
# minigzip is smaller than gzip.
progs minigzip # 0KB.
ln minigzip gzip
progs cp      # 0KB.
progs rm
progs ls
progs kill
progs df      # 0KB.
```

```
progs ps      # 4KB.
# ns is a lightweight version of netstat.
progs ns      # 4KB.
ln ns netstat
progs vm      # 0KB.
progs cat     # 0KB.
progs test    # 0KB.
ln test [
progs hostname   # 0KB.
progs login   # 4KB.
progs getty   # 4KB.
progs stty    # 4KB.
progs w       # 0KB.
# uptime gives you only the first line of w's output.
ln w uptime
# msg is a lightweight version of dmesg.
progs msg     # 0KB.
ln msg dmesg
progs kget    # 0KB.
progs reboot  # 0KB.
# less is smaller than more
progs less    # 36KB
ln less more

# sysctl is a program for changing kernel variables;
# it's needed, for instance, to enable IP Forwarding.
progs sysctl
progs swapon  # 0KB.
progs pwd_mkdb  # 0KB.
progs dev_mkdb  # 0KB.
progs umount
progs mount_std

progs route      # 8KB
# If you need an editor ee is as small as they get although it is
# at least debatable why you would need an editor in the firewall.
#progs ee       # 32KB.
#libs -lncurses

# It might be useful to have tcpdump for debugging purposes.
progs tcpdump    # 100KB.
special tcpdump srcdir /usr/src/usr.sbin/tcpdump/tcpdump

progs arp # 0KB.
# I wouldn't NFS mount anything on a firewall, but it can be done.
#progs mount_nfs    # 0KB.
#ln mount_nfs nfs
progs ping    # 4KB.
#progs routed  # 32KB.
progs traceroute  # 0KB.
ln mount_std procfs
ln mount_std mount_procfs

# It's nice to be able to ssh into your firewall and see how it's doing.
progs sshd    # includes ssh and scp

# These programs are needed to control IPFilter.
progs ipf ipfstat ipnat ipmon
```

```
# IPFilter logs using syslog which should be configured to log remotely to a
centralized log server.
progs syslogd

progs chflags

# Libraries Needed
libs -ledit -lutil -lmd -lcrypt -lmp -lgmp -lm -lkvm
libs -lmytinfo -lipx -lz -lpcap -lwrap
libs -ltermcap -lgnuregex -ltelnet
libs -lcrypto
```

The process of selecting the programs for the floppy is basically a trial and error procedure. You think of something you would like to have in the floppy, you add it, the image turns out to be too big, you think again if you really, really need it (or delete something else), and so on.

As you can see in the crunch.conf example, the firewall should have as few programs as possible to accomplish its job. You could strip this list even further, but don't make the system completely unusable. You should still be able to login to it and troubleshoot it.

The two directories mfs_tree and floppy.tree will be copied to the MFS and floppy file systems, respectively. You should have there the minimum set of configuration files needed for the system to function properly.

In my mfs_tree directory I only have a /etc directory containing a stripped down version of the following files:

disktab	host.conf	profile	services
fstab	hosts	protocols	shells
gettytab	login.conf	rc	termcap
group	motd	remote	ttys

The rc file is a special script in PicoBSD. While rc is called by init, just like in any other UNIX, the shell script overwrites itself during execution. The reason for this is that no file created in the MFS can be modified without recompiling the kernel. Although you can modify them in a running system, the changes will be lost if you reboot the machine. By having a minimal rc in the MFS that only copies the configuration files from the floppy and rewrites itself, we can achieve the most flexibility for system configuration. The real rc in the floppy file system can be modified by mounting the floppy on another machine, and it will not be necessary to rebuild either the kernel or the floppy binary image.

The rc script file that is in the MFS will be something like this:

```
#!/bin/sh
### Special setup for one floppy PICOBSD ###
# WARNING!!! We overwrite this file during execution with a new rc file.
# Awful things happen if this file's size is > 1024B

stty status '^T'
trap : 2
trap : 3

HOME=/; export HOME
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin
export PATH
dev="/dev/fd0c" #
trap "echo 'Reboot interrupted'; exit 1" 3
```

While rc is called by init . . . the shell script overwrites itself during execution.

```
# Copy from MFS version of the files, and then from FS version.
echo "Reading /etc from ${dev}..."
mount -o rdonly ${dev} /fd
cd /fd; cp -Rp etc root / ; cd / ; umount /fd
cd /etc
#rm files to stop overwrite warning
for i in *.gz; do
      if [ -f ${i%.gz} ]; then
      rm ${i%.gz}
      fi
done
gzip -d *.gz
pwd_mkdb -p ./master.passwd
echo "Ok. (Now you can remove ${dev} if you like)"
echo " "
. rc
exit 0
```

In the file system floppy, you should put the configuration files that you expect to change. For instance, you can put your firewall rules (ipf.conf) and your NAT rules (ipnat.conf) there if you're doing NAT. Another file that is usually there is the master.passwd file used to generate the passwd and db hashes used for authentication.

Here are the files I have in my floppy.tree /etc directory:

```
ipf.conf    ipnat.conf      rc      sshd_config
master.passwd     resolv.conf     syslog.conf
```

The interesting file here is rc. This rc script is the one that overwrites the rc in the MFS file system. This file is where I configure the network interfaces, load the firewall rules, load the NAT rules, start the appropriate daemons like SSH and syslogd, and rebuild the password files.

```
#!/bin/sh
mount -a -t nonfs
rm -rf /var/run/*
hostname firewall
ifconfig lo0 inet 127.0.0.1 netmask 0xff000000 up
ifconfig xl0 inet XX.XX.XX.XX netmask 0xffffff00 up
ifconfig xl1 inet YY.YY.YY.YY netmask 0xffffff00 up
route add default ZZ.ZZ.ZZ.ZZ
route add -net 192.168.0.0 192.168.0.1
ipf -Fa -f /etc/ipf.conf
ipnat -FC -f /etc/ipnat.conf
sysctl -w net.inet.ip.forwarding=1
(cd /var/run && { cp /dev/null utmp; chmod 644 utmp; })
sshd -f /etc/sshd_config
syslogd -s
dev_mkdb
cat /etc/motd
exit 0
```

Since this file is in the floppy file system, changing it is very easy. You only need to mount the floppy on any UNIX machine and make any modifications you want. Since the rc file in the MFS overwrites itself with this rc file, your modifications will have an effect on the firewall host.

Another trick is to put a /root directory in the floppy tree with a .ssh/ subdirectory. You can then store your SSH public keys in the floppy file system and configure SSHD to use only public key authentication.

After you have all files ready, it's only a matter of running the PicoBSD configuration script. The script is in the directory build/ and it's called picobsd. It has a menu which lets you modify various parameters and build the binary image of your floppy.

Once you have the floppy image done, you can use dd(1) to transfer it to a real floppy and boot your firewall from it. Remember to format the floppy first to make sure it does not have any bad blocks.

Once you have your one floppy firewall operational, there are some things you can try experimenting with. One of them is the kernel run levels in FreeBSD, and thus PicoBSD.

There is a variable in the FreeBSD kernel that specifies in which security context the kernel should operate. The name of the variable is kern.securelevel, and the default is -1 which means that no security is enabled. The possible values are:

-1 Permanently insecure mode – always run the system in level 0 mode. This is the default initial value.

0 Insecure mode – immutable and append-only flags may be turned off. All devices may be read or written subject to their permissions.

1 Secure mode – the system immutable and system append-only flags may not be turned off; disks for mounted file systems, /dev/mem, and /dev/kmem may not be opened for writing; kernel modules (see kld(4)) may not be loaded or unloaded.

2 Highly secure mode – same as secure mode, plus disks may not be opened for writing (except by mount(2)) whether mounted or not. This level precludes tampering with file systems by unmounting them, but also inhibits running newfs(8) while the system is multi-user.
In addition, kernel time changes are restricted to less than or equal to one second. Attempts to change the time by more than this will log the message "Time adjustment clamped to +1 second."

3 Network secure mode – same as highly secure mode, plus IP packet filter rules (see ipfw(8) and ipfirewall(4)) cannot be changed and dummynet(4) configuration cannot be adjusted.

The MFS and floppy file systems could be built with all files set as immutable, and right after loading the firewall rules you could switch to run level 3. The only disadvantage of this is that once the system is running you can no longer change anything, but I think that's as secure as you can get with a firewall.

Conclusion

Although firewalls are usually built using general purpose operating systems, they do not need all of the programs and utilities that come by default. All that firewalls really need are the kernel and a couple of programs for configuring network interfaces, loading the rules, etc. We have seen that all of these programs fit nicely in a single 1.44MB 3.5" floppy disk. There is no reason to have all of the extra unused programs in the disk, even if there is enough space for them. They make it possible to accidentally turn on an unwanted service. They also give a trespasser a rich development environment from which to launch further attacks. I believe that eliminating all of these unused programs makes a firewall more secure. In the case of a break-in, it gives the attacker an almost unusable system from which no further attack is possible.

. . . once the system is running you can no longer change anything, but I think that's as secure as you can get with a firewall.

a crash course in managing security

by David Brumley

David Brumley is well known for his site *http://www.theorygroup.com* and for his role as Assistant Computer Security Officer for Stanford University.

dbrumley@stanford.edu

Introduction

This article is about managing infrastructure from a computer security perspective. In this article there will be three recurring themes. The first theme is the need to do things the proper way the first time. A correctly written program will always act predictably, even on bad or malicious input, and hence be secure. Similalrly, when designing an infrastructure each service is reduced to its essential components. Each component is combined to act predictably.

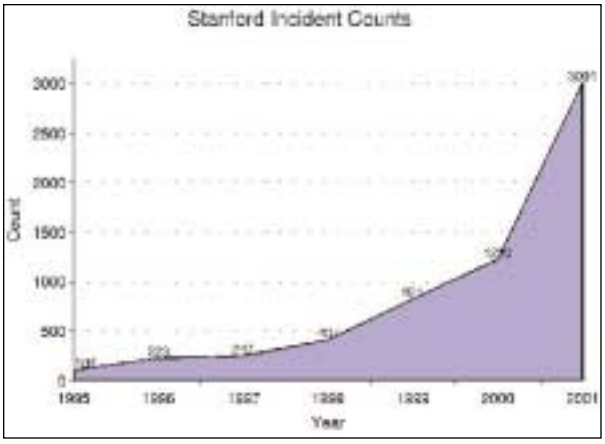
The second theme is the need for proper planning. A clear and consistent plan keeps all the different parts of the computer infrastructure functioning together. Without a plan, security is a catch-up game, constantly behind the latest exploits and news. With a clear plan, security provides *defense in depth*. As new components are added to the infrastructure, defense in depth will be enhanced, not compromised. By considering security implications at the appropriate levels of infrastructure planning, a system can become not only more secure but also easier to use and more robust, and can provide better availability.

The third theme is that scalable infrastructures can reduce overall cost while enhancing security. A scalable infrastructure is one that not only can be extended, but also can decrease the cost of extension. Scalable infrastructures are modular, allowing new technology to update and extend the old.

The Need

Do your users have any expectation of privacy? Do you have assets that need protecting? Have you considered the cost of a system compromise? These are dumb questions. Yet, we still fool ourselves into thinking we aren't the targets.

A picture is worth a thousand words.



This graph depicts the incident counts from Stanford University. Included in the incident counts are attempted intrusions from one of our network links to the Internet.

Notice the figure depicts an exponential increase in incidents. Each year, except one, the incident count doubled. These figures demonstrate that computer security incidents are on the rise. Without automated, scalable mechanisms to resist attacks, systems will be compromised quickly. For example, the recent "Code Red" Internet worm was able to compromise some servers at Stanford within only hours of installation.

Now think back to the questions I just asked about computer security. These incident counts are alarming because users expect some level of privacy and protection on the Internet, which you must provide.

What Is Computer Security

The term computer security often conjures up a mental image of a teenage hacker breaking into the US Department of Defense computers. In a dimly lit room investigators watch a screen "trace back" the "hacker," with a SWAT team on hand for good measure. While this sounds exciting, it is not what computer security is primarily about.

Computer security is the art of ensuring confidentiality, integrity, and availability of compute resources.

CONFIDENTIALITY

Confidentiality is the best understood of the three areas. Every computer has something that needs to be kept private. On one end, there are databases containing the crown jewels: intellectual property. On another end, there are user passwords and other personal information.

Even systems which don't contain sensitive data may have access to one of the largest pools of data at your organization: the network. Network sniffers can gather enough information to leave all your systems vulnerable.

For example, in February 2001 several computers at Stanford were compromised. These computers were used by faculty, primarily for email. Nothing *on* the computer was confidential. However, using the computers, the hacker who broke in gleaned confidential information from the network the computers were on.

```
xxxxxx.Stanford.EDU => yyyyyy.Stanford.EDU [110]
USER user1
PASS password1
STAT
UIDL
QUIT

———- [FIN]

xxxxxx.Stanford.EDU => yyyyyy.Stanford.EDU [23]
 !'''#P
38400,38400#xxxxxxx.stanford.edu:0'DISPLAYxxxxxxx.stanford.edu:0XTE
R
Muser2
password2
elm
jjjjjjjjjjjjjjjjjjjj
———- [Timed Out]

xxxxxx.Stanford.EDU => yyyyyy.Stanford.EDU [21]
(#USER user3
PASS password3
SYST
PORT 171,65,0,0,5,104
LIST
CWD /home/pub/gary
CWD /home/pub/
CWD /home/

———- [Timed Out]
```

While I have removed the actual usernames and passwords, be assured they were just as clear as the words "user" and "password" are above.

The total log file the hacker had of passwords was approximately 8,000 lines long. The point is that just because a system doesn't have direct access to confidential data, it is quite likely it has indirect access to confidential data via the network.

ENSURING CONFIDENTIALITY

Confidentiality is best ensured by using encryption. Any data that is sensitive should be encrypted.

Network sniffers can gather enough information to leave all your systems vulnerable.

Some of the most important data to encrypt is user authentication tokens. Those who still use clear-text telnet and FTP should immediately develop a plan to switch to a model that protects the authentication data from eavesdroppers.

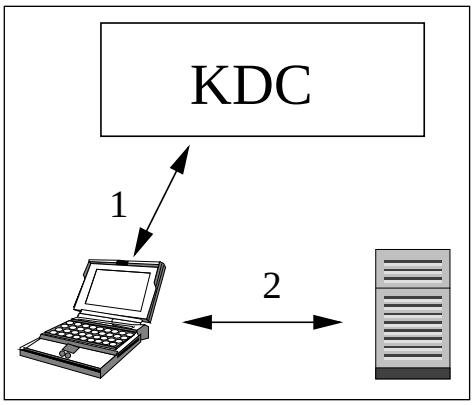
Think back to the cracker's sniffer log file I showed you earlier. Encrypted communication turns those clear-text logins into a jumbled mess of ASCII characters. In that same file was the following entry:

```
xxxxxx.Stanford.EDU => yyyyyy.Stanford.EDU [23]
%%user1%IR.STANFORD.EDU@(P^$:-)':ca<'%.+vc6s}DF~T[f8FLc|vI;#wG\CN6MYI
P%6M-&&&&
& #'$&&Y`&&VT100&
wI\cfCCSDK) >aWHW^H
>rGhsN{q0jxU
`&$$ vQa;j:T8%H>VzL d>7s_
——- [Timed Out]
```

The difference is clear. Those who used clear-text authentication such as telnet and FTP had to change their passwords. In most organizations, this would result in a help desk call. Those who used an encrypted protocol did not have to change their password, saving a help desk call.

This is just one example of how computer security measures save money over the long term.

Universities generally have a large user base that changes rapidly. Stanford, like many, uses Kerberos as our base authentication model. Kerberos provides a central infrastructure for managing user information.



Kerberos uses a Key Distribution Center (KDC), which contains authentication information on all users. (The following is a simplified description of Kerberos. For a more accurate report, please see the reference.) Using Kerberos, a user contacts the KDC for an authentication token (called a ticket). The KDC sends back an authentication token encrypted with the user's password. If the user can type in the password correctly, they can decrypt their authentication token. This is step 1 in the figure.

The user can use the decrypted token (ticket) to authenticate to other services (step 2).

The centralized model gives:

- A single place for adding new users
- A single authentication scheme for adding new services
- An abstraction between service and authenticating to that service

If we need to disable a user, we can do so in the KDC. Further, once disabled in the KDC the user cannot use *any* service. We don't have to go from the email service to the file server to the Web server and revoke login privileges . . . it's all done in one place.

At Stanford, our KDC contains over 58,000 active principles. Every year, each new student is given a principle. Each graduating student has their principle suspended. All of this is done automatically, without manual processes, just as computers should.

Integrity

The integrity of compute resources generally relies upon proper enforcement of protection domains, such as file permissions. When a system is compromised, the cracker can do anything. Besides installing an Ethernet sniffer to grab unencrypted network chatter, as above, the cracker can replace system utilities to hide his presence.

If you don't stop the initial compromise, there is a good chance with a skilled attacker that you will never notice the intruder. You simply won't see the bad guy's changes to the system. For example, a hacker can replace ps so you do not see their processes. They can replace ls so you do not see the files. They can install a kernel module so you cannot see their open files.

ENSURING INTEGRITY

There are two axioms (attributed to Cheswick and Bellovin) when *hardening* a system against attack:

- all programs are buggy;
- if a program isn't run, it doesn't matter if it is buggy.

In essence, these principles mean each computer should ideally only run one service. That service should run with the economy of a mechanism utilizing the principle of least privilege on a stripped down operating system.

Unneeded services and programs should be removed because:

- Programs periodically need to be patched, and fewer programs means less patching, an often time-consuming task.
- More resources are available to the needed service.
- Unneeded services add complexity to the system.
- There will be fewer services to support.

Unfortunately OS vendors ship computers with most services enabled. They do this on purpose: they want the computer to be easy to use. They also assume that the user will customize the system appropriately.

Pushing out secure servers to the end user means disabling all programs that are not needed. At Stanford, we created SULinux to do just that.

SULinux is secure. There have been no recorded compromises of an SULinux system. More importantly, SULinux gives us the opportunity to not just secure the system, but also to integrate the host into our environment. Users don't just see the security, they see the increased usability.

There are approximately five installations of SULinux per day on average, or about 2,000 per year. Assume a low-ball estimate that each compromise costs the university approximately $300 to fix. Costs include downtime and employee time to reinstall, regardless of whether research or other data was modified. Given the current rate of scanning, it's appropriate to assume any unpatched system would be compromised at some time or another.

Thus, the savings can be estimated at 2,000 hosts x $300 = $600,000 per year. This is only a ballpark figure, but it demonstrates the scale of the problem and the possible savings from implementing security solutions.

Most operating systems allow for automated installation. By distributing hardened versions of the OS, whether it be through Windows RIS or Ghost images or IRIX roboinst, you can significantly increase the security and integrity of compute resources, while at the same time allowing users to easily integrate into the infrastructure provided by central IT.

AVAILABILITY

The goal for computer services is 99.999% availability. Computer vendors tout the number, yet realizing the 5 9's in the real world is difficult.

There have been no recorded compromises of an SULinux system.

The company was caught with their proverbial pants down.

Achieving 99.999% availability in practice requires technology that will work consistently. Remember that a secure program is a correct one, and it will produce an answer or error whenever possible. Availability is built upon programs functioning in exactly this manner. Hence, a highly available service must be built upon secure components.

Today, there are many threats to availability, including system intrusions, DoS attacks, and service hijacking.

Computer security can increase availability by keeping hackers from compromising systems, creating robust services that resist DoS attacks, and properly securing your domain to prevent hijacking. Though computer security alone can't accomplish each of these things, it can facilitate such an environment when implemented in a consistent and serious manner.

For example, one of the leading companies in computer security had their Web site defaced in February 2000, right before the widespread DDoS attacks. The hacker changed the Web site by compromising the DNS server, pointing their main splash page to point to a Web server in Brazil. The company was caught with their proverbial pants down. Their site was defaced for over 13 hours, along with significant downtime of their primary DNS server while they fixed the problem.

The reason: a hacker named Dennis Moran, age 18, who lived across the country on the east coast. The lesson here is security can not only allow the organization to save face, but it can also help minimize downtime. In this specific case, a coherent security plan may have:

- Prevented the system compromise.
- Detected the system compromise.
- Planned for a backup server to minimize downtime in case of compromise.
- Provided staff so that compromises could be reported and resolved quickly and easily.

Each would have reduced downtime. Each must be planned and implemented before the compromise.

How to Start

There are consultants charging well over $500/hour who will help you implement a security architecture at your organization. If you need a consultant to overcome political boundaries, by all means hire one.

If you don't have the cash, I'll tell you for free what to do. First, create a position or office for a person who will be in charge of computer security at your organization. At Stanford that office is called the Computer Security Office. Having a central authority is the only way I know of for security to be effectively implemented in any organization. Authority over computer security should be given to the person responsible for it. This may sound trite, but it is the most often overlooked aspect of creating a sound infrastructure. (A corollary: if you are ever offered a security position without authority, run away.)

Next, you should find the correct people to be responsible for computer security in the organization. The correct person will understand computer security risks and be able to evaluate them with a long-term perspective. Those without a long-term perspective generally do not last.

Last, you should let the people do their job. If you have given security the proper position and authorization, and you have selected the right people, the rest will take care of itself. This is because security is a process, not a goal.

This isn't just what you should do, this is how Stanford implemented the Computer Security Office. Our policies give the office responsibility, and we have hired the best talent around.

Policies

Effective policies are essential to the success of an organization. Foremost, policies should give the community the expectations surrounding the use of compute resources. While policies often differentiate between acceptable and unacceptable behavior, their purpose should be to let the community know how the compute resources are to be used to realize the goals of the organization.

For example, at Stanford the goal of the organization is learning. Our compute resources are for academic pursuits. Our policy explicitly mentions that only incidental non-academic use is tolerated.

Second, policies provide a consistent mechanism for addressing the eventual security incident. More, carefully crafted policies and procedures provide a chain of command so that incidents can be dealt with quickly, efficiently, thoroughly, and consistently.

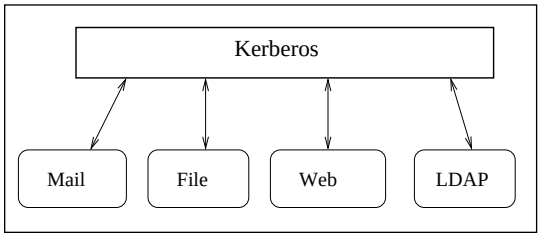
At Stanford, the computer-use policy appoints the computer security officer at the top of the chain, with administrators and then users following.

The Plan

At a high level, the critical areas to plan well are

- What base-level authentication system to use
- How to ensure system integrity
- Educating the community on using that infrastructure
- Educating the community on security-related matters

The answer Stanford has come up with is we will use Kerberos for base-level authentication. Authorization is handled through the local services, such as AFS file permissions.



Everything hinges on our authentication choice. Kerberos is used to authenticate for file access (AFS), Web access (WebAuth), directory services (LDAP), and even enrolling in classes (AXESS)

Ensuring system integrity is not complete. Our partial solution includes SULinux for Linux, Norton Anti-Virus for Windows, Best Use Documents for other OSes, and periodic vulnerability assessments.

The Computer Security Office, along with others, educates the community. We give talks, presentations, and provide documentation and tools to the community.

Quick Response

On May 29, 1999, over 30,000 people received a hate mail derogatory to certain racial groups. The mail was forged to appear to be from a Stanford engineering student, probably in order to exact some sort of revenge.

USEFUL URLS.
SULinux – *http://sulinux.stanford.edu*

Stanford Security Office – *http://security.stanford.edu*

Stanford Kerberos Infrastructure – *http://lelandsystems.stanford.edu/services/kerberos/*

David's Site – *http://www.theorygroup.com*

Over 30,000 people believed that this student sent the mail. Within an hour angry people were outside the innocent student's residence. The situation was critical.

Without proper resources the incident could have turned into a hefty political problem. However, Stanford was able to act quickly because the Computer Security Office was already in place to handle this sort of problem. Within 14 hours of the hate mail, the Security Office was able to identify suspects and distribute a response from the university president. The quick response turned a potentially disastrous situation into a positive racial-awareness campaign.

While not everyone will have a similar hate-mail incident, every organization will at some time need leadership during a computer security crisis. It may be an Internet worm or a DoS attack. Having a group for computer security ensures a quick response.

As a Public Service

In February 1999, a computer intruder who went by the nickname "ShadowKnight" compromised several Stanford computers. Some of the computers compromised were responsible for one of NASA's largest and longest-running research projects, with a multi-million-dollar budget.

On November 6, 2000, Jason Diekman, aka "ShadowKnight," appeared before a United States district judge and pleaded guilty to recklessly causing damage to a protected system, unauthorized access of a non-public computer, and unauthorized use of an access device.

Our office coordinated the investigation with several Stanford network administrators. The network administrators were capable of assisting because of a commitment at all levels to providing a safe and secure Internet. The results of that investigation were turned over to the FBI, who then arrested and prosecuted Mr. Diekman.

Protecting People

A CAT scan uses radiation to map out the human body. A CAT scanner is a computer, normally running SGI's IRIX operating system.

IRIX machines by default have a "guest" username with no password. These same IRIX boxes with the "guest" account are used in CAT scan machines. Imagine a hacker simply logging into your local medical facilities computer and viewing or changing the results of this diagnostic procedure.

Critical infrastructure, such as the CAT scan machine, should be identified. After identification, regular audits protect the organization from liability. More, audits alert people to potential problems before they affect the people who use the equipment daily.

Summary

Creating a security infrastructure is just like planning, implementing, and deploying any other service. While you may receive more recognition for your deployment of a Web mail service, the long-term safety of the community is considerable compensation. Though many may be angry when they must change the way they compute, you are giving the community good habits that will follow them into careers outside the university. Finally, implementing a coherent computer security strategy may protect you, your organization, and your coworkers from harm done by computer intruders.

no plaintext passwords

by Abe Singer

Abe Singer is a computer security manager at the San Diego Supercomputer Center, and occasional consultant and expert witness. His current work is in security measurement and security "for the life of the Republic."

abe@SDSC.EDU

Introduction

Compromise of a user password is one of the most difficult intrusions to detect. Historically it has been difficult or impossible to avoid transmission of passwords in the clear. But the technology now exists to make this possible, albeit not trivially. The San Diego Supercomputer Center (SDSC) has managed to eliminate plaintext password transmission, while continuing to deliver services to a widely distributed user base. While it took some technical effort, overcoming the human hurdles proved to be more challenging. This article discusses what solutions we provided and how we managed to do it without annoying too many people. We have actually added value to the environment, instead of reducing it.

At SDSC, we have to deal with some interesting issues of scale. We have thousands of users and very few support staff. We have a wide variety of operating systems, high-speed networks and high-performance storage systems. Our users expect to be able to move large amounts of data (terabytes) around, in a reasonable amount of time. They want to do streaming applications, grid computing, and stuff that has not yet been invented. In addition to providing computing resources to researchers, we have people doing research in high-performance computing, networking, and storage. Unlike many places, most of our users do not work inside networks that we control. They are spread all over the planet and work for different institutions. This means that our infrastructure must scale outside of our "trusted" networks.

Because of the nature of our users, and the work done within and outside the Center, we cannot (and do not want to) mandate homogeneity such as "everyone must use Outlook for email." Instead, we focus on supporting protocols, and let the users pick their clients. We attempt to provide reasonable support for the applications that our users are already using, instead of requiring them to use the one(s) that we've decided are easy to support. Oh, and by the way, we have not had a root-level compromise (that we are aware of) on our managed systems in over two years.

How do we do it? Mostly through the following:

* Strong configuration management
* Patch early, patch often
* Strong authentication, and no plaintext passwords, anywhere
* Simple, but strong, access control between "trusted" and "untrusted" networks

We have managed to turn off plaintext passwords and continue to provide support for almost all of the applications our users have. Additionally, we will provide services for applications that we don't support. For instance, we provide IMAP over SSL service, and support Netscape Mail and Outlook clients. However, if a user has another client that speaks IMAPS, they are welcome to use it. We just won't help them with problems with their client.

The result is that we have an environment where users can get their work done from anywhere in the world. They can use the software that they need and read their email with the clients that they like, and we have improved the security of our systems at the same time.

. . . most switches behave like hubs when their MAC tables are overloaded

Background

Most of the commonly used TCP protocols use plaintext passwords: telnet, the r-commands, ftp, pop, imap, and HTTP basic authentication. Other protocols can use either plaintext or encrypted passwords but may use plaintext passwords by default.

Effective access control requires strong authentication. This means using authentication mechanisms which cannot be easily bypassed or subverted through eavesdropping, cryptanalysis, or brute-force attack.[1]

An authentication scheme that is highly resistant to brute-force attack or cryptanalysis can be fundamentally useless if the password can be intercepted. Protection of passwords on hosts is reasonably well implemented: both UNIX and Windows provide encrypted storage of passwords and prevent exposure of the passwords to the users. However, many network protocols transmit these same passwords in plaintext.

Why is this a problem? Primarily because plaintext passwords can be easily intercepted via a sniffer. There are dozens of sniffer programs available.[2] Some sniffers are smart enough to filter out just the usernames and passwords, and produce username, remote host, and password in an easy-to-read format.

As mentioned above, we have not had a root-level compromise on our managed systems in over two years. But we do have some networks with systems managed by users or other groups. We have had compromises on those systems, and occasionally we help investigate intrusions on other systems. Most of the intrusions we've seen include the use of a sniffer.

An intruder may have any number of motives for breaking into a system: running an IRB "bot," setting up a "warez" site, or using the system as a cutout to attack other systems, for example. The intruder typically installs a rootkit, and the rootkit almost always includes a sniffer. Even when sniffing is not an intruder's primary motive, the sniffer is an opportunistic attempt to compromise user accounts on other systems. Since users often use the same password on multiple systems, an intruder will try the username and password on various machines, even at different sites, to see what they can log into.

In one case, a user burned passwords to three different sites, including ours. The user had set up their own system (on the "user-managed" network) because they supposedly needed to run their own FTP server. They would routinely telnet into the system, and from there ssh into our site and the two others. Eventually their system was compromised (due to an unpatched vulnerability). The intruder used the system to run a bot but also installed a sniffer. When we found the sniffer log, we saw several usernames and passwords into multiple sites. We notified the other sites involved and investigated our managed machines to determine whether or not the intruder had actually used the passwords (apparently not).

"Switched" networks are not immune to sniffing. Switches sometimes leak information. Some switches are not fully switched but are really "switching hubs," where groups of ports share data exactly like a hub. Most importantly, most switches behave like hubs when their MAC tables are overloaded.[3]

One of the big problems with password compromises is that they are difficult to detect. Since the intruder logs in with a legitimate username and password, they are successfully authenticated and look like a legitimate user to the system. A user account compromise can go undetected for months – in one case we know of, a compromise

went undetected for two years! Some detection is possible using user profiling, but this is cumbersome and inaccurate. We believe that efforts are better spent at eliminating the opportunity for interception in the first place.

The more effective solution is to either encrypt the password in transmission or authenticate without password transmission.

The Rollout

About three years ago, SDSC began turning off most plaintext password services. A year ago we turned off the last, with the exception of a few older systems that we haven't yet updated. (The long time frame was partly due to a lack of technology and partly due to the need to make sure that users were able to make the transition.) These systems allow plaintext within our trusted networks, but not outside.

We began by enabling SSH and Kerberos services. Users had the option of using Kerberized clients or SSH. FTP was enabled via tunneled SSH sessions. We bought 5,000 copies of SecureCRT and several hundred copies of F-Secure SSH for Macintosh to distribute to users who needed it.

We then announced that plaintext access would be cut off in nine months. We notified all user via email and mentioned the change in the message-of-the-day and in banners. All the messages included links to Web pages for information on how to get software and how to use it. We also periodically sent out reminders.

On the scheduled date, we turned off access to Telnet, rlogin, and FTP. We did this by changing TCP-wrappers to deny access and display a banner with an explanation and a URL for more information. Most of our users had already switched. Some of them had not, but as soon as they tried to log in and saw the rejection message, they had little choice but to make the appropriate transition. Very few called our help desk, as they sheepishly realized that we had given them plenty of notice. A few (20 or so) did call. Most of them had some problems understanding. One user, when asked if he had read seen our notices, responded, "I never read those things. They never say anything useful."

One user, when asked if he had read seen our notices, responded, "I never read those things. They never say anything useful"

We implemented email solutions as they became available. About a year and a half ago, we drew a matrix of all the email clients we had to support, the non-plaintext authentication mechanisms they supported, and servers implementing the same. We found that we could turn off plaintext access to email using a combination of IMAPS, POPS, APOP, KPOP, and NFS access for mail-reading from managed UNIX systems. We also found a Web-mail solution (IMHO Webmail)[4] to provide users with access to their email from any SSL-capable Web browser.

Six months later, after appropriate announcements and lead time, we turned off plaintext email services.

We have also rolled out SecureFTP,[5] sftp, and are evaluating Web-based access to user directories.[6]

The Technology

We describe here the various solutions we have implemented, with some tips based on our experience. We are not providing a tutorial on how to implement POP or IMAP servers. Rather, we discuss what we use to encrypt passwords on these services. References are provided for details of implementation.

Kerberos is a mature, well-reviewed, open protocol, having been around about 15 years.

Interactive Access

For interactive access, we support Kerberos[7] and SSH.[8]

Kerberos works very well. Installing and configuring Kerberos is not trivial, but it is very easy to use. Kerberos provides strong authentication (both user and host) without transmitting passwords, and can provide encrypted data transmission. It scales very well (our KDCs are Sparc5s), has low overhead, and provides redundancy for failover and remote administration tools. Kerberos is a mature, well-reviewed, open protocol, having been around about 15 years. See note 7 for a detailed source of information on running and installing Kerberos.

SSH is a replacement for Telnet which provides interactive shell access, rcp-like file copying, and the ability to tunnel other protocols across encrypted streams. All data streams in SSH are encrypted.

We are currently supporting version 1 SSH because, until recently, there were not version 2 clients for all of the platforms we have to support (Windows, UNIX/Linux, Macintosh). We support Kerberos authentication for SSH sessions.

We also support RSA public keys for authentication via SSH. This requires a user to generate a public-key pair, store the private key on their client machine(s), and install the public key on the server(s). We have mixed feelings about this option, as it requires users to keep their private key secure, and users are not known for being good at keeping data secure.

A source for information on various SSH clients can be found in reference 8.

Email

We support a variety of services for email. Our users have Eudora, Netscape Mail, Outlook, and others.

Our users have to be able to read and send mail from any location. However, we do not want to be an open relay for the entire world. Our solution involves authenticated SMTP over SSL, IMAP over SSL (IMAPS), APOP, POP XTND XMIT, HTTPS, POP over SSL, and KPOP.[9]

Here is what we support:

	Reading Mail		*Sending Mail*	
Client	Protocol	*Daemon*	*Protocol*	*Daemon*
Eudora	APOP	qpopper	POP XTND XMIT	Sendmail, qpopper
Outlook	IMAP/SSL	UW imapd, sslwrap	AUTH SMTP/SSL	Sendmail
Netscape	IMAP/SSL	UW imapd, sslwrap	AUTH SMTP/SSL	Sendmail
Webmail	HTTPS, IMAP	Roxen,IMHO, imapd	HTTPS	Sendmail, Roxen, IMHO
Mutt, Elm, Pine	NFS		SMTP	
Eudora	KPOP	qpopper		
Outlook	POP/SSL	qpopper, sslwrap		

IMAPS and POPS are implemented using sslwrap.[10] This is almost trivial to do.

We have a centralized mail hub running Sendmail. We will not relay mail for machines outside of our network, without authentication. However, users outside our network who need to send mail have the option of using either authenticated SMTP over SSL, or the XTND XMIT option of POP. Netscape Mail and Outlook support the former, Eudora supports the latter.

For UNIX mail clients (pine, elm, mutt) on internal hosts, we provide NFS access to incoming mail folders. We only allow NFS on "trusted" networks, which are the networks which only have hosts that we manage. Since users have to use Kerberos or SSH to access these hosts, they have already used a strong authentication method to access the system.

sslwrap is a relatively simple way to encrypt a TCP-based service. To sslwrap a service, first configure the service to accept connections only on the loopback interface (127.0.0.1). This can be done easily with TCP-wrappers (and you should be TCP-wrapping your services anyway). Next, place an entry in inetd for the secure version of that service (e.g., IMAP uses port 143, IMAPS uses port 993).[11]

The inetd.conf entry for imap and imaps looks like this:

```
imap    stream  tcp nowait  root    /usr/local/etc/tcpd  /usr/local/etc/imapd
imaps   stream  tcp nowait  nobody  /usr/local/etc/tcpd  /usr/local/etc/imapsd
```

imapsd is a simple shell script that looks like this:

```
/usr/local/etc/sslwrap -cert /usr/local/certs/ssl-imap.pem \
  -CAfile /usr/local/certs/ca-cert.pem -port 143
```

The TCP-wrapper configuration for these services looks like this:

```
imapd: 127.0.0.1: allow
imapd: ALL: rfc931: DENY
imapsd : ALL : rfc931 : ALLOW
```

The ssl-wrapper negotiates an encrypted session on the "secure" port, then connects through the loopback device to the original unencrypted service. The remote client gets an encrypted session, and the local client does not require any modification. The same can be done to implement POPS with any POP server.

KPOP is not trivial to configure, and Eudora only supports version 4 of Kerberos. The server must be compiled with Kerberos libraries, and is run on an alternate port with a command line option to enable Kerberos authentication. We have not had many users making use of KPOP. Refer to the qpopper documentation for instructions on implementation.[12]

APOP uses a challenge-response password hashing mechanism to avoid transmitting passwords in the clear. When a client connects to the server, the server presents a challenge string. The client hashes the user's password with that challenge, and returns the hash. The server authenticates the client by comparing that hash with its own hash of the password. In order to implement APOP, qpopper has to keep clear-text copies of user passwords in its own database, by default /etc/pop.auth. Tools are provided for managing this password database.

The XTND XMIT feature of qpopper allows mail delivery through the pop server. The pop server in turn delivers the mail by calling Sendmail locally. This allows us to pre-

sslwrap is a relatively simple way to encrypt a TCP-based service.

. . . the scp protocol has a two-gigabyte file limit, which is problematic for some of our users

vent open mail relaying on our mail server and still enable users to send mail through their POP clients.

XTMD XMIT also has to be configured on the client. For Windows Eudora clients, this requires editing a .ini file. For Macintosh clients, it requires that the Esoteric Settings plug-in be installed.[13]

Authenticated SMTP over SSL implements the SMTP AUTH feature,[14] using SSL to encrypt the data stream. We implement this feature using Open Sendmail,[15] with SASL[16] and the entropy-gathering daemon.[17] With Sendmail, the server must be dedicated to authenticated SMTP, so we run a separate mail host just for authenticated relaying.[18] A source for detailed implementation instructions is in this reference.

For users who insist on using other mail clients which do not support the above protocols, we also support SSH-port forwarding.[19] Our POP and IMAP servers allow plaintext authentication via the loopback device. A user can establish an SSH connection to our mail server and then tunnel any mail client they want. SSH tunneling can be tricky to do, but it does work.

Finally, we have some users who may not have access to a mail client. They may be using another person's machine, in a terminal room in a conference, or at an "Internet cafe," for example. In addition to the various client support above, we provide a Web-based mail client called IMHO (see note 4). The client runs under the Roxen[20] Web server, and talks IMAP to the mail server via localhost.

File Transfer

For file transfers, we support scp through SSH (version 1), sftp (through SSH version 2), KFTP, and SecureFTP (see note 5).

scp is supported through the SSHD server. However, the scp protocol has a two-gigabyte file limit, which is problematic for some of our users.

sftp uses a separate binary which is invoked by the SSHD server. An entry in the SSHD configuration file points to the sftp binary.

KFTP is a Kerberized version of FTP. It uses Kerberos authentication on the command channel, but data transfers remain plaintext. This is considered a feature by some of our users.

Many of our users are not concerned with data confidentiality, only data integrity. Most are researchers using open or published data. Many of them transfer large amounts of data – sometimes terabytes. In these cases, the overhead of encryption creates too large a performance problem. At one point in time, our users found that encryption increased transmission time by a factor of four.

SecureFTP is an ssl-wrapped FTP server, with command line clients and a Java-based client that can be run from a Web browser. The command channel is encrypted, which protects passwords during transmission. Like KFTP, the data channel is left unencrypted. Any FTP server can be ssl-wrapped, and the Java client can be run from any operating system (which supports Java). See reference 5 for where to find details.

File Sharing

We currently do not allow file sharing outside of our trusted networks. Within our networks, we provide NFS for UNIX systems, Netbios/SMB for Windows systems, and

AppleTalk for Macintosh. We have a handful of centralized file servers where all user data lives – home directories, project areas, etc.

NFS does not provide user authentication. We only export to trusted hosts, and the NFS server will only talk to trusted networks.

We use Samba to provide file sharing for the Windows systems. We do this so that we can export the same data to the Windows systems as we do the UNIX systems. Samba can authenticate using UNIX passwords, its own password file, or through a Windows PDC, but password encryption is only available for the latter two methods. We authenticate Samba users against a PDC, which is also used to authenticate Windows logins. In order to match file ownership properly against a UNIX system, Samba requires that the UNIX usernames match the Windows usernames, or you must manually maintain an equivalence list.

We use Netatalk[21] to provide AppleTalk file shares similar to how we provide Windows file shares. Currently this uses plaintext passwords, and is restricted to our Macintosh networks.

Other

We also maintain an anonymous-only FTP site. It is configured anonymous-only so that users aren't tempted to use the server for file transfers. All users have an incoming and outgoing folder that they can use anonymously. This provides a fallback method of transferring files when authenticated access is unavailable.

PGP software is available for those who wish to use it. We provide version 2 and version 5+ software.[22]

Weaknesses

Our system is not (yet :-]) perfect. There are some known weaknesses, most of which will be addressed over time.

Our biggest problem is what we call "two-hopping." A user at a remote site, who does not have an SSH client on their desk (machine A), will telnet to another system which does have SSH (machine B), and then SSH into our site. The password into our site is intercepted by a sniffer between A and B. The intruder then uses the password to SSH into our site. Sometimes the less-than-clueful user telnets through several machines before ssh-ing into ours. In some cases, we have alerted the user and site administration, and changed their passwords, only to have it happen again a few days later. When asked to install SSH, the user complains that it is too hard, or they don't have the fifteen dollars for a site-licensed copy! (and now there are free versions of SSH available)

Our biggest problem is what we call "two-hopping."

Macintosh file sharing via AppleTalk currently sends passwords in plaintext. As mentioned above, we expect to move to DoubleTalk and Samba.

Several of the services we implement (e.g., APOP and Kerberos) require access to stored plaintext passwords (encrypted on disk with a shared key). While this is less than desirable, the hosts on which these passwords are stored are within our control and are kept relatively secure. The risk to us is much less than a user storing a plaintext password on their home computer.

The anonymous-only feature of our FTP server (wu-ftpd) does not reject the login until after the password is provided. The result is that users who don't know the server is anonymous-only will "burn" their password the first time they try to use the service.

REFERENCES

1. B. Schneier, *Applied Cryptography*, 2nd ed. (John Wiley and Sons, Inc., 1996).

2. SecurityFocus: sniffers, http://www.securityfocus.com/templates/tools_category.html?category=4&platform=-&path=|%20sniffers%20|

3. S. Sipes, "Why Your Switched Network Isn't Secure," http://www.sans.org/newlook/resources/IDFAQ/switched_network.htm, The SANS Institute, September 10, 2000.

4. S. Wallström, B. Lincoln, IMHO Webmail, *http://www.lysator.liu.se/~stewa/IMHO*.

5. G. Cohen, B. Knight, SecureFTP, *http://secureftp.sdsc.edu*, 2000.

6. Y. Last, WebRFM, http://www.geocities.com/SiliconValley/Horizon/7772/webrfm.html, 1999.

7. "Kerberos, the Network Authentication Protocol," *http://web.mit.edu/kerberos/www/*, September 10, 2000.

8. "OpenSSH for Windows and Mac", *http://www.openssh.org/windows.html*, July 25, 2001.

9. M. Crispin, RFC 2060, "Internet Message Access Protocol – Version 4rev1," December 1996; J. Myers, M. Rose, RFC 1939, "Post Office Protocol – Version 3," May 1996; A. Freier, P. Karlton, P. Kocher, "The SSL Protocol Version 3.0," *http://home.netscape.com/eng/ssl3/draft302.txt*, November 18, 1996.

10. R. Kaseguma, sslwrap, *http://www.rickk.com/sslwrap/*, 1999.

11. Protocol Numbers and Assignment Services, *http://www.iana.org/numbers.html*, Internet Assigned Numbers Authority, April 30, 2001.

12. qpopper, *http://www.eudora.com/qpopper/*.

13. "Email FAQ," *http://www.netgate.net/html/email_faq.html*; "Changing POP (or other) Port in Eudora," *http://www.eudora.com/techsupport/kb/1501hq.html.*

14. J. Myers, RFC 2554, "SMTP Service Extension for Authentication," March 1999.

15. Sendmail, *http://www.sendmail.org*.

16. J. Myers, RFC 2222, "Simple Authentication and Security Layer (SASL)," October 1997.

17. EGD, *http://www.lothar.com/tech/crypto/*.

Windows password encryption is known to be weak.[23] We only use this protocol on internal networks and limit it to Windows-only networks.

We are also vulnerable to keystroke sniffers. An intruder at a remote site can install a keystroke sniffer and intercept passwords as they are typed. In our experience, keystroke sniffers are rather rare, and our risk is relatively low. The only solution to this would be one-time passwords or hardware tokens, which for us is not worth the expense.

Policy

It probably goes without saying that it is important to have policy behind your technology.

As always, support from management is instrumental. Our management takes security seriously and has supported our efforts. What has helped us gain that support is showing that we are enabling services, as opposed to denying access. Additionally, we make sure that management knows what we are doing, and is comfortable using the technology, before switching off services. In this way, when the disgruntled big-ego researcher calls the director to complain about not being able to use Telnet, the response is, "I'm able to use it, why can't you?"

We sometimes appeal to ego to encourage recalcitrant users. For instance, when dealing with a researcher, first we'll find out who their biggest rival is. The conversation then goes something like this: "Well, Dr. X, Dr. Y [the rival] had no problem installing and using SSH. Perhaps we could ask one of his grad students to come over and show you how to use it." This works more often than you might think.

The other key strategy is to give users plenty of advance warning. We typically give six months' to a year's warning, with email reminders, items in the message-of-the-day, and banners on services. Even with all this notice, some users will not get the information. So we make sure that help-desk staff are prepared to support them. In some cases, we preemptively help out users who we know will have difficulty.

Clients that store plaintext passwords for the convenience of the user are problematic. An intruder on a remote machine can pluck these passwords out of the files where they are stored. We don't have any (technological) way to prevent users from using these features. We do ban user storage of plaintext passwords on our managed systems.

Gotchas and Issues

When configuring any encrypted service, first make sure that authentication works properly without encryption. It can be easy to assume that there is a problem with session encryption when the real problem is that authentication is failing.

When enabling encryption, verify that the transmitted passwords are actually encrypted. Some services can be easily misconfigured, so that you think that passwords are encrypted when they actually are being sent in plaintext.

Password distribution/management is not trivial. All of these systems require that we manage user passwords on a variety of systems. We do not use a centralized account management service (e.g., NIS), because most of them are insecure and/or don't work with all of our systems.

We have a home-grown Web-based password changing system, which sets UNIX passwords, Kerberos pass-phrases, APOP passwords, and Windows passwords. We do not

manage all passwords from a centralized database but explode the passwords out to their respective systems. It's not the best system, but it works for us.

Be aware of software that people can install on their desktops on their own, such as VNC (it can be SSH-wrapped), personal Web servers, FTP servers, etc. We ban these as a matter of policy, but it is difficult to prevent.

Future Directions

We eventually will replace /bin/login with the Kerberized version. This version will log users in using their Kerberos pass-phrase, and get a ticket-granting ticket all in one shot. This will allow us to use the KDC as the central password management system for UNIX logins. We would like to integrate Windows 2000 into this environment, but its feasibility remains to be seen.

We are evaluating a Web-based system for users to access our file systems (WebRFM, see note 6), providing the ability to upload and download files through an encrypted, authenticated site. The system looks promising.

We have also started looking at OpenAFS[24] with Kerberos. AFS provides true user-authenticated file sharing and can be used effectively for file sharing between different sites.

Conclusion

Due to the ubiquitous use of sniffers, disabling plaintext passwords is critical for effective protection of systems. There is no single solution that provides universal access, but effective service can be provided through a combination of technologies, policy, and careful user handling.

Thanks to the following people who were involved in the actual implementation: Tom Guptil, Tom Perrine, Jeff Makey, Cindy Zheng, Haisong Cai, and Dave Savilonis.

For additional documentation see *http://security.sdsc.edu/self-help/no-plaintext/*.

18. B. Bannister, "Implementing Authenticated SMTP with Sendmail," *http://security.sdsc.edu/publications/smtp-auth.shtml*.

19. "Port Forwarding," *http://www.ssh.com/products/ssh/administrator30/Port_Forwarding.html*, May 2001.

20. Roxen Web Server, *http://www.roxen.com/products/webserver/*.

21. Netatalk, *http://netatalk.sourceforge.net/*.

22. Pretty Good Privacy, *http://web.mit.edu/network/pgp.html*.

23. l0phtCrack, *http://www.atstake.com/research/lc3*.

24. OpenAFS, *http://www.openafs.org/*.

the bookworm

BOOKS REVIEWED IN THIS COLUMN

CYBERREGS

Bill Zoellick

Boston: Addison-Wesley, 2001.
Pp. 307. ISBN 0-201-72230-5.

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 is Editorial Director at Matrix.net. He owns neither a dog nor a

peter@matrix.net

I'm writing this on September 17. That means that the issue of security is very different from the one I've written about for more than 15 years.

In the '50s and '60s (and early '70s), computer security meant, in general, the physical security of the multimillion-dollar mainframe. The advent of the DEC 10, the Multics project, and UNIX yielded password security questions. The ARPANET introduced other questions, carefully and wittily discussed by Bob Metcalfe in RFC 602 (December 1973), which I still recommend.

Cryptography, firewalls, etc., help us somewhat where data security is concerned, but I fear that an analogy with our homes or cars is apt. We lock our doors; in some neighborhoods, folks have bars on their windows, burglar alarm systems, "kryptonite" on their bicycles, and "The Club" on their steering wheels.

None of these will avail where a determined attack is concerned, just as moats and armor offered deterrents and shields, not absolute protection.

The news is full of statements that "the world has changed" as of Tuesday morning (Eastern Standard Time). Our sense of security and safety has changed. That's for sure.

On September 14, the FBI sent an advisory to InfraGard members, who are mandated with keeping the nation's digital infrastructure intact, warning them to upgrade their security precautions in light of recent terrorist activity. But InfraGard comprises federal agencies, which is hardly a significant segment of the Net.

The effect of the devastating terrorist attacks of September 11 have caused businesses and the government to act in anticipation of cyber-assaults.

In a hearing on Wednesday, September 12, the U.S. Senate Governmental Affairs Committee detailed how prone critical-systems computer networks are to cyberterrorism. They concluded that security measures taken where government systems were concerned was poor.

Etc.

Crackers, viruses, DDoS attacks, idiots with backhoes, morons running trawlers in the China Sea, earthquakes, hurricanes – that's what we generally concern ourselves with. Not with terrorist activity.

I was in Copenhagen for uptime(1), the celebration of UNIX's 10^9 second, and got home after midnight on September 11. I got into my office after about six hours' sleep to find everyone clustered around the TV set.

Just terrible.

And a book, too.

Bill Zoellick has written a fine book about the old version of the universe. Perhaps, after a while it will be relevant to the new Internet world, too.

Zoellick has written a couple of other books, and I've always found him reliable. The volume at hand is a thoughtful essay on the ways that government regulation may impinge on business where the Internet and the Web are vital.

My problem is that I fear that government regulation and intrusion will increase, making Zoellick's notions inapplicable.

It's a thought-provoking read, but events may have overtaken it.

book reviews

PROGRAMMING RUBY: THE PRAGMATIC PROGRAMMER'S GUIDE

DAVID THOMAS, ANDREW HUNT ET AL.

Boston: Addison-Wesley, 2001. Pp. 564.
ISBN 0-201-71089-7.

Reviewed by Raymond M. Schneider
ray@hackfoo.net

For those who do not already know, Ruby is a programming language from Japan. It has been suggested that Ruby is more popular than Python. The Pragmatic Programmers have tackled the task of providing the world with the first book documenting this language.

As any good programmer knows, you need the right tool for the right job. Well, Ruby is another excellent language to add to the toolbox. The book is broken down into five sections: Facets of Ruby, Ruby in Its Setting, Ruby Crystallized, Ruby Library Reference, and Appendices.

The Facets of Ruby section begins by quickly introducing the prospective Ruby programmer to the basics. It covers all of the usual suspects: arrays, hashes, and control structures. For those Perl regular expression folks, you'll be happy to know that Ruby's syntax for regexs is much like Perl. The remaining chapters in this section of *Programming Ruby* provide more depth on the various things Ruby has in common with object-oriented programming languages and completes the brief overview of Ruby's syntax and language characteristics.

Ruby in Its Setting shows some of what's in store for someone utilizing Ruby in various scenarios. The first chapter of this section covers Ruby's command-line syntax and flags. There are no surprises here. The following two chapters, Ruby and the Web and Ruby TK, are fun with plenty of examples to play with.

Ruby Crystallized, the third section, is an even more in-depth look at dealing with Ruby's semantics. This section has more information on the behavior of classes and objects in Ruby, looking at safe levels and tainting as well.

The Ruby Library Reference section is just that, designed for those of us who do not feel the need to read a book on programming in yet another language but just want a reference. This section of the book (starting on page 279 and going through to the Appendices) is just excellent, not only providing explanations of what things are but also including many examples of the way things should be implemented.

This book is excellent in its approach, whether you like to read through the basics with each language you learn or just want to tackle the references. It should satisfy you no matter what sort of reader you are.

Lastly, this book has actually been made available under the Open Publications License. It is on the Web, and you can download it at: *http://www.rubycentral.com/book/index.html*. Happy Ruby Hacking!

THE OPERA 5.X BOOK

J.S. LYSTER

San Francisco: No Starch, 2001.

Reviewed by Rick Leir

We have seen IE clobber Netscape in the browser market, and many companies now don't much care whether their Web applications or Web sites work with anything but IE. This sets the stage for the "big company in the Northwest" to dominate the Web as it does the desktop. I am interested in any browser which has the merits to buck the trend. Opera might have a chance.

Dominate? There are Web sites which only work in IE, forcing you to have a desktop machine with a Microsoft OS. Most people have such a machine available to them, so Web designers tend to cater to IE users, and make use of the corresponding Microsoft server software. I sense a vicious circle here.

What browser should you use? IE has a user interface (UI) that I just don't like. Netscape sadly lost its war and is now being sent in the wrong direction by AOL. Mozilla 9.1 is good for the knowledgeable Linux or *nix user. Amaya is the W3C example browser. Konqueror is a KDE/Linux project. Opera is an excellent alternative. I like its UI, speed, size, and availability for lots of OSes (MS Windows, Solaris, OS/2, Macintosh, Linux [x86, SPARC, PowerPC], EPOC, and BeOS).

Opera is a free browser with ads. Turn off the ads for a (low) purchase price. If you as a professional expect your browser to be free of cost, go soak your head.

New users: This book will be useful to anyone new to Opera, even if they are new to Web browsers in general. It is said that you can divide users of an application into categories: beginner (10%), average (80%), and advanced (10%).

This book will help a beginner quickly get into the average user category and will encourage the average user to make the step to advanced. Sysadmins who provide Opera on their systems will want to have a few copies of this book to make available to users.

Web developers: If you want a nice, standards-compliant browser, Opera is a good choice, and you will find this book useful for Opera configuration. If you have to develop for both IE and Netscape, you may find it easier to develop for Opera and then test in IE and Netscape since Opera is more compliant to standards.

This book starts with a good description of the UI: all menus, buttons, and options. There are some subtleties here, such as an option to pretend to be the IE

book reviews

browser so certain sites don't go into a "lowest common denominator mode" or reject you completely. Keyboard shortcuts are presented, and they can speed you up considerably. The book continues with a description of mail and newsgroup access that will help even people who are new to Usenet.

Next is a description of Java and plugin configuration, which is important because Opera defaults to minimal plugins so it can be lightweight.

For more advanced users, there is a very brief introduction to CSS so that Opera styles can be configured and a very brief introduction to security protocols so that cookies and certificates will make more sense. An introduction to HTML leads into using Opera in full-screen mode for presentations or kiosks.

Up to this point ,the book focuses on Opera 5 for Windows, but it follows with two chapters on the differences for other OSes.

There is an attached CD containing versions of Opera for various OSes, plugins, and sample HTML/CSS code. This will save you hours if you don't have a broadband link.

To wrap it up, there is a "Brief Contents" before a "Contents in Detail," which together make it easier to get around, and an index.

This book is typeset with a nice choice of fonts in a casual style. Look for a purple cover with a conductor's baton!

USENIX news

Criticality and USENIX

by Daniel Geer

President, USENIX Board of Directors

geer@usenix.org

September 11th hit close to home. I write this as a patriot of civilization. I hope that's possible, because that is my premise. I don't want to argue, which I mean in the deeply emotional sense of the word "want."

Harris Miller, President of the Information Technology Association of America, testifying before the US House Government Subcommittee on Government Efficiency, Financial Management, and Information Technology, very eloquently said: "Many people are unsure what homeland defense means and unclear on how they can participate. I would like to suggest an immediate action: safeguard US computer assets by adopting much more widely sound information security practices."

I'd like to endorse that but as a citizen of civilization more than of the United States, which you should not misinterpret as a diminution of my first allegiance.

All of us in USENIX like to say that the Net has no borders. Some of us call themselves Netizens. A few think that no borders means that they are outside of every (and hence subject to no) civil authority; more know that no borders instead means that they are inside every civil authority, that everyone's rules apply rather than no one's. In that sense, the Net is now the carrier of civilization and you can no more roll that back than command the wind. It is not a digital divide; it is a civilization divide.

In the United States, business in aggregate spent $100 billion ($10^{11}$) on the Y2K issue, and business did it as a risk management activity goosed along by civil authority. On a worldwide basis, the figure was somewhere between $300 billion and $800 billion. Compare that, if you will, with a total worldwide spend in calendar 2001 for cyber security that is estimated to be no more than $10 billion. Civilization has yet to comprehend that on the Internet it is still September 10th.

Speaking for security people everywhere, we have gotten what we asked for (the spotlight) and now the real work begins. We are faced with fantastic demand pull and an expectation of miracle-working. Side effects are likely. A hard design requirement (inescapable) plus a hard design constraint (indetectable) will challenge us, but economic trade-offs favor broad implementation for the first time in our careers. "Lead, follow, or get out of the way" has never meant as much. If you, as do I, think of security as a proper subset of reliability, then getting the Net from September 10 to September 12 means attention to the matters that USENIX people are all about, whether that is systems management, operating system resilience, programming languages more likely to help than hurt, and so forth. We are the crème de la crème of what it takes. We, that is You, have to lead, follow, or get out of the way; I recommend lead.

Wealth comes from productivity and nowhere else. Our productivity is ever more dependent on our electronic infrastructure. More to the point, our productivity gains have to come from or through our electronic infrastructure or

USENIX MEMBER BENEFITS

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

Free subscription to *;login:*, the Association's magazine, published eight times a year, featuring technical articles, system administration articles, tips and techniques, practical columns on security, Tcl, Perl, Java, and operating systems, book and software reviews, summaries of sessions at USENIX conferences, and reports on various standards activities.

Access to *;login:* online from October 1997 to last month *http://www.usenix.org/publications/login/login.html*.

Access to papers from the USENIX Conferences online starting with 1993 *http://www.usenix.org/publications/library/index.html*.

The right to vote on matters affecting the Association, its bylaws, election of its directors and officers.

Optional membership in SAGE, the System Administrators Guild.

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/membership.html

OR CONTACT

office@usenix.org

Phone: 510 528 8649

they are not going to come. If you like wealth, then act like you do and pay real attention to the one central fly in the ointment: On the Internet, every sociopath is your next door neighbor. That makes attacks inevitable unless, of course, the uncivilized cannot grasp the centrality of the Net and thus overlook it as the target of choice. Not a bet I would take. Not something we can treat as a wait-and-see experiment either since, unlike in a tank battle, in the electronic sphere the cost for the defense is 100x the cost for the offense and it is that magnification of the hostile and the insignificant that make the Net, and our role in and around it, the critical playing field.

Your patriotic duty is to act in proportion to the extent that the computer systems and technologies you influence are of any value beyond personal playthings. Wanna be thought part of the global economy? Smell the global coffee. Want the citizenry to depend on you as an integral part of a global infrastructure? Act like it matters and lock the damned door. Think risk management is a game for cool heads and ruthless assessment? Take a deep breath of discipline.

If the genie were standing here asking me for my wish, then I'd wish for the word USENIX to be always used in this sentence: "When it matters you find a USENIX member," sweetening that with "and it matters more than ever." In some sense, our biggest ally is the insurance industry who will, you can be assured, start rating digital risks as closely as they rate flame-front spread speeds in draperies.

All this requires knowledge and I am asking you to help me review if not revise how to pick and choose both the knowledge that we need and the knowledge that we need to propagete. In some sense, USENIX has always done a better job of intellectual property detection than Patent Offices have done at protection, but just because something is novel and non-obvious does not mean it is valuable. Our first forays into distance learning have just begun which now seems prescient as travel permissions start to tighten. We will have to shrink some meetings to make others work. What can we do to make that genie unnecessary? Help us think. Help us help you do. Lead.

IETF Network Management People Meet LISA Network/System Administrators

by Bert Wijnen

Area Director for IETF Operations and Management Area

bwijnen@lucent.com

[The note below is, in my mind, what conferences are all about. If you're a network manager going to LISA, please consider attending this BOF. The Editor]

There will be a BOF at LISA 2001 on Wednesday evening running from 7pm-11pm, the purpose of which is to start a dialog between the Internet Engineering Task Force people working on network management technologies and network/system administrators who actually run enterprise networks. The goal is to better understand the needs of the network/system administrators and to evaluate how IETF technologies can help to get the job done and to identify what is missing. The IETF is especially interested to receive input from people who are running large enterprise networks.

The IETF NM folk have already had sessions with NANOG (North America) and RIPE (Europe) ISP operators. From these meetings, a set of requirements have been gathered and described in: *http://www.ietf.org/internet-drafts/draft-ops-operator-req-mgmt-00.txt* (By the time you read this, there may be a revision 01).

Some people think that the Enterprise network operators have different requirments. So it would be good to read the above document and evaluate if your requirements are included, and if not, then to make sure that we (IETF NM protocol folk) get your input.

EXPLORE THE FUTURE OF SYSTEMS ADMINISTRATION AT THE 15TH SYSTEMS ADMINISTRATION CONFERENCE

LISA 2001

December 2–7, 2001

San Diego, California

INVITED SPEAKERS INCLUDE:

GREG BEAR, several times Hugo and Nebula winner,
Best selling author of *EON, SLANT & DARWIN'S RADIO*
"SLIME VS. SILICON--LIFE'S A BITCH,
BUT WOULD YOU WANT TO BE A COMPUTER?"

ERNEST PRABHAKAR, Apple
"REBUILDING THE DIGITAL ENTERPRISE
AROUND INTERNET STANDARDS"

WILLIAM LEFEBVRE, CNN Internet Technologies - JUST ADDED!
"CNN.COM: FACING A WORLD CRISIS"

PHIL COX, SystemExperts Corp.
"HARDENING WINDOWS 2000"

JOHN S. FLOWERS, nCircle Network Security
"ARMORING THE NEXT WAVE OF SECURITY TECHNOLOGY"

LISA 2001 ALSO OFFERS OVER 50 TUTORIALS, HALF OF WHICH ARE NEW INCLUDING:

- Real-World Intrusion Detection
- Network Security Profiles
- Hacking-Exposed: LIVE!
- An Introduction to Computer Security NEW!
- Practical Wireless IP Security and Connectivity
- Creating a Computer Security Incident Response Team NEW!
- UNIX Security Threats and Solutions NEW!
- Forensic Computing

ATTENDEES RECEIVE A FREE COPY OF *SELECTED PAPERS IN SYSTEM ADMINISTRATION*, TO BE PUBLISHED BY WILEY IN DECEMBER AND EDITED BY ERIC ANDERSON, MARK BURGESS, AND ALVA COUCH

http://www.usenix.org/events/lisa2001

RECEIVE AN ADDITIONAL DISCOUNT
WHEN YOU REGISTER ONLINE!

Sponsored by
USENIX The Advanced Computing Systems Association
SAGE The System Administrators Guild

CONNECT WITH USENIX & SAGE

MEMBERSHIP, PUBLICATIONS, AND CONFERENCES

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 SITES

<http://www.usenix.org>

<http://www.sage.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 part or in its entirety requires the permission of the Association and the author(s).

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

;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