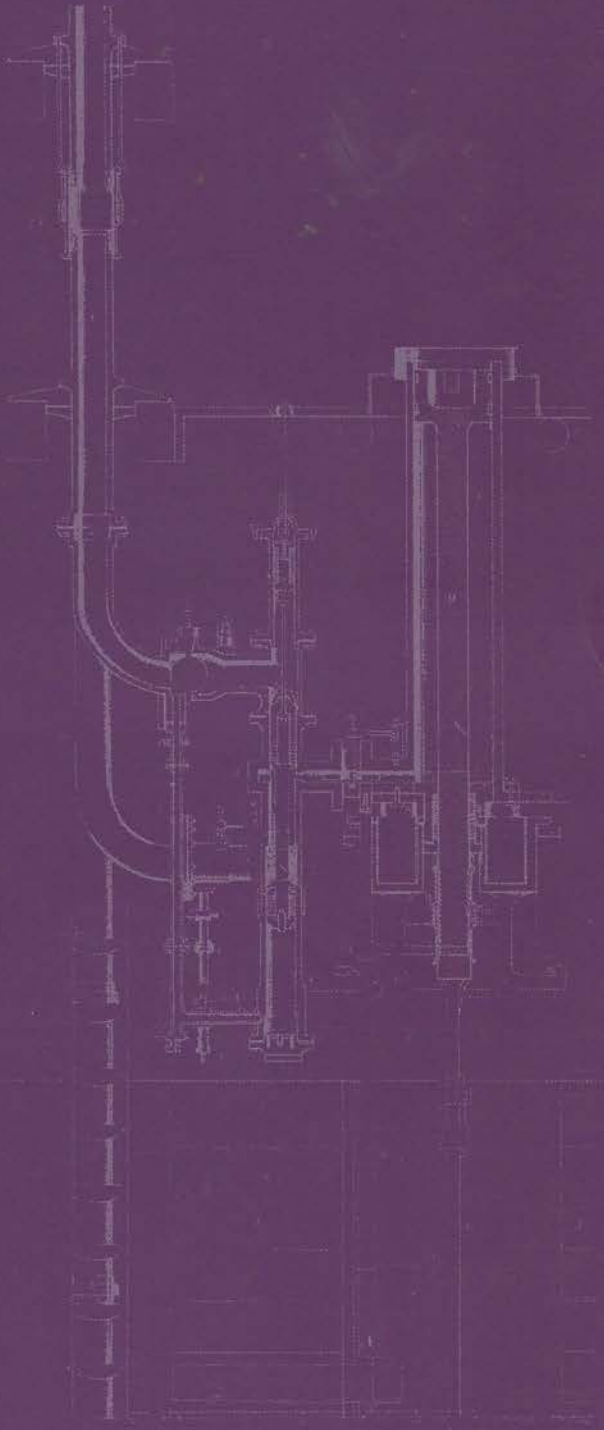
DECEMBER 2004 VOLUME 29 NUMBER 6

;login:



USENIX

USENIX Upcoming Events

2005 USENIX Annual Technical Conference

APRIL 10–15, 2005, ANAHEIM, CA, USA
http://www.usenix.org/usenix05

2nd Symposium on Networked Systems Design and Implementation (NSDI '05)

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

MAY 2–4, 2005, BOSTON, MA, USA
http://www.usenix.org/nsdi05

3rd International Conference on Mobile Systems, Applications, and Services (MobiSys '05)

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

JUNE 6–8, 2005, SEATTLE, WA, USA
http://www.usenix.org/mobisys05

10th Workshop on Hot Topics in Operating Systems (HotOS X)

JUNE 12–15, 2005, SANTA FE, NM, USA
http://www.usenix.org/hotos05
Paper titles and abstracts due: February 1, 2005
Full paper submissions due: May 2, 2005

14th USENIX Security Symposium (Security '05)

AUGUST 1–5, BALTIMORE, MD, USA
http://www.usenix.org/sec05
Paper submissions due: February 4, 2005

19th Large Installation System Administration Conference (LISA '05)

Sponsored by USENIX and SAGE

DECEMBER 4–9, 2005, SAN DIEGO, CA, USA
http://www.usenix.org/lisa05

4th USENIX Conference on File and Storage Technologies (FAST '05)

DECEMBER 14–16, 2005, SAN FRANCISCO, CA, USA
http://www.usenix.org/fast05

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

;login:

VOL. 29, NO. 6, DECEMBER 2004

EDITOR
Rik Farrow
rik@usenix.org

CONTRIBUTING EDITOR
Tina Darmohray
tmd@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PROOFREADER
proofshop
proofshop@usenix.org

TYPESETTER
Star Type
startype@comcast.net

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

office@usenix.org
login@usenix.org
conference@usenix.org

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.

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

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

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

letters to the editor

DEAR EDITOR,

Rik Farrow writes in the August *;login:* that "the downside of [offshoring] is that real communication between software developers and program managers will get even worse." This presumes a model where developers only communicate with managers, not directly with users. My advice for how to become offshoring-proof is to educate yourself in contextual design (or any of the other methodologies that likewise presumes extensive contact between developers and users), and to complement your computing skills with a liberal-arts education, which will develop your communication skills and your ability to understand noncomputing perspectives.

MAX HAILPERIN

max@gustavus.edu
http://www.gustavus.edu/+max/

Wireless Security: A Discussion

Note from Rob Kolstad: This article summarizes an email discussion between Marcus Ranum and Bill Cheswick after Marcus had an "interesting experience" at the USENIX Security Conference. Thanks to both of them for allowing us to share it publicly in order to foster discussion.

MARCUS RANUM:

I had an interesting experience at the USENIX Security Conference, and I'd like to share it here for discussion. Like many conference attendees, I took advantage of the wireless network so I could check my email, update my Web site, etc. At virtually every USENIX conference, someone sets up dsniff and collects passwords as they cross the wireless, and this latest conference was no exception. For the past few years I've basically chosen to ignore the snoopers because, frankly, I hoped they'd grow up and go away. This year I finally got sick and tired of it, and confronted one of the snoopers who had emailed me my own password.

What bothered me most about this experience was that the folks who do the snooping are security practitioners. When I raised the issue, the immediate response was surprising. Basically, I got the exact same set of excuses that crackers have been using for years: "I wasn't abusing the information," "it was for my own research/curiosity," etc. I'm afraid I lost my temper quite badly in the face of what seemed to me to be a lack of clarity on the part of the security community regarding basic issues such as whether or not we're justified in doing exactly the same things as the "bad guys" as long as we're the "good guys." I think the whole situation was further exacerbated by the fact that the whole issue was not in my opinion taken adequately seriously by the USENIX Board members at the conference.

So I think the whole incident becomes a microcosm of today's security experience. It motivates me to ask questions like:

- What is the difference between the good guys and the bad guys if their actions are largely the same?
- Why do we place the onus of self-defense on the victim, instead of demanding ethical behavior from the perpetrator?

I felt that my privacy was being violated, and, more to the point, I was going to be forced to waste time installing security measures because of someone's "harmless curiosity." Indeed, I find it ironic (if not outright contradictory) that USENIX, which is normally a haven for privacy advocates, would tolerate this behavior over a lengthy period of time.

BILL CHESWICK:

I just returned from the San Diego USENIX Security Conference, reliably one of the top security conferences of the year; this year's was no exception. The keynote in particular was one of the best security talks I have heard in years (and I hear a lot of security talks); there were several excellent papers; and most of the hall track meetings alone were worth the trip.

I had several things to accomplish at this conference, including preparation for an invited talk I was asked to give at the last moment. These activities were curtailed when I was accused of being legally and ethically on the wrong side concerning the use of the dsniff program.

The incident precipitated swirls of hallway conversations about the legalities and ethics of using dsniff. This is not your average crowd of I-am-not-a-lawyer-buts as they debated the ethics and legalities of password sniffing. Not only has this crowd assisted in numerous law enforcement cases, many have advised lawyers, courts, and the US Congress on such matters.

Much of what many of us have done is "ahead of the law" (to quote one lawyer), and since Leviticus (and Numbers, adds Dan Geer) and the New Testament appear to be mute on the topic, we have traditionally had to grope our own way towards personal and societal rules for using the Internet. Concerning the legality of using dsniff, the IANALBs appeared to cover the spectrum from "illegal" to "not covered." Several laws appear to be involved, and it looks like a courtroom toss-up to me.

I have given a lot of thought to the ethics of various Internet experiments and practices I have adopted over the years. I believe that this is an especially important thing to do, given this novel medium and the nascent state of case law. I have used sniffed passwords to make an important point in a number of my talks in the past. I am still satisfied with the ethics of doing so, despite the lecture I received, but the point is not important enough to fight for. I have agreed not to display sniffed passwords publicly, for any reason, in the future, and did not do so at the invited talk.

I think USENIX's response was measured and proper. They asked us nicely not to sniff the network, and I, for one, complied. That is about all they can do without closing the network or implementing extremely invasive procedures. I expect that future conferences will include similar requests.

Marcus raises several specific issues. Some are matters of basic law; the rest deal with our community's position on the forefront of a new technology.

Marcus asks, "Are individuals justified in doing exactly the same things as the 'bad guys' as long as we're the 'good guys'?" The problem here is in the question. We are not doing "the same things as the "bad guys." The bad guys break into systems, compromise their integrity, modify their software, etc.

Marcus asks, "What is the difference between the good guys and the bad guys if their actions are largely the same?" I think "largely the same" is neither ethically nor morally "the same." The crackers who justify their actions in court with "I wasn't abusing the information" and "it was for my own research/curiosity" aren't in court because they ran dsniff.

Even though this was an upsetting experience for me, the hallway track continues to provide deep, thoughtful discussions on the cutting-edge issues of our industry and society.

MARCUS RANUM:

Bill observes that many security practitioners are "ahead of the law," but I feel that professional conduct, and what is tolerated by an organization like USENIX, should be about "right and wrong." Hiding behind legalisms is not leadership. USENIX is an organization full of privacy advocates, an organization that cares enough about its members' privacy that it protects attendee email addresses and the like. By consistently turning a blind eye to people sniffing the conference network, USENIX has implicitly encouraged the kind of "anything goes" attitude that is more appropriate at DEFCON than at a respected conference concerned with its attendees' privacy. As an organization of thought leaders in the computing arena, I think USENIX should pay attention to the leadership shown by conferences like SANS, which will eject attendees for sniffing the conference WAN or any other hacking-type activity. If we are, indeed, "ahead of the law," then it's more important that our behavior be, literally, exemplary.

I'm a fairly open person, and I've always been interested in helping other practitioners with their research. If Bill had wanted to know how often I change my password, he could have just asked. Instead, he stole what would have been gladly given, and was amused by the fact that he was able to. It's almost a USENIX tradition that some wiseacre posts passwords on the bulletin board with a sign saying "CHANGE THESE" at every conference. This whole issue would never have surfaced if Bill had asked for, and gotten, permission from USENIX to sniff the network before doing so. That opens the broader question of whether such permission would have been granted. I doubt it—but it would have been easy to find out. I suspect we're dealing with a case of "better to ask forgiveness than permission." I've already forgiven Bill, and I hope he's forgiven me for screaming at him in public; I think it's good that this issue has been aired before USENIX has to deal with an incident involving less forgiving people.

BILL CHESWICK:

As for asking permission, that's quite true, and I have done so at other conferences, with the explicit purpose of reporting the penetration of secure protocols into the common packet stream. Of course, I never cared about the particular passwords that were sniffable, Marcus's or any others, and I only forwarded his results to him as a friendly nudge. Some previous nudges had resulted in the discovery of a non-functioning encrypted tunnel, and I have been thanked for my efforts on those occasions.

Would permission have been granted? I never thought that deeply about this in this particular venue: I skipped the step in my rush to deal with other pressing things at the conference. I should not have.

I have also let bygones be bygones. I'm glad we got this out in the open.

USENIX RESPONDS:

This has been an instructive experience for all of us. Future conference directories will indeed attempt to give a more explicit description both of acceptable behavior and of the risks inherent in use of the wireless network. We thank both Marcus and Bill for their mature and measured discussion of these issues.

RIK FARROW

editor's thoughts (a.k.a. musings)

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

rik@spirit.com

WELCOME TO THE SIXTH SECURITY edition of ;login:. Like past editions, this one contains articles about security topics that I consider to be among the most important current issues. I want to thank the authors who wrote for this edition. It is the authors who provide the content. All I do is find them and cajole them into writing.

This issue also includes the summaries of the 13th USENIX Security Symposium.

The Security Symposium started out with what some people considered a depressing keynote. Earl Boebert, of Sandia National Laboratories, and one of the authors of Multics, provided a rather pessimistic look at operating system security. I tend to agree with a lot of Boebert's assertions, and you can read them for yourself in the summaries.

You can also read Jonathan Shapiro's response to the keynote. Shapiro points out that there were some very good reasons why Multics did not succeed in the marketplace, outside of the "crap in a hurry" that Boebert mentioned in his keynote speech. Again, you can read Shapiro's thoughts on this topic for yourself.

I have often written about the failures of operating systems in my Musings columns. Largely based on the goals of Multics, operating systems were, at least in theory, supposed to protect a system against poorly written software—that is, logic or programming faults in software should never compromise the security of a system. The operating system should encapsulate the faulty process and prevent software flaws from changing the overall state of the system. As we all know, this is not how operating systems work.

Instead, what we see are systems that are exploited via a process that displays email for a user,or ones that permit a non-privileged user to become a privileged one, and totally compromise the security of a system.

I believe there really are two questions to ask about the future security of our operating systems. First, is it possible to build an operating system that is really secure and can be used by anyone? Second, do we have the will to end the current fiasco, and actually begin using secure operating systems?

Some people might argue that today's operating systems are secure. The Linux Security Module in the 2.6 series of kernels does provide hooks for adding much more comprehensive access controls than exist without the LSM. LSM does provide support for more control, but at the cost of complexity. These same hooks appear in FreeBSD 5, and come with the same level of complexity. OpenBSD takes a different approach, based on profiling processes and systrace.

All BSD versions support the `jail()` system call for isolating processes in a changed root environment that includes process isolation and some control over the network environment as well. But a proper jail setup also means proper firewall configuration. The jail only becomes reliable at the network level in cooperation with configuration that is external to the jail. The jail also does not prevent against CPU DoS attacks, because it does nothing about process scheduling (read Kirk McKusick's article in the August 2004 issue of *;login:* for more about BSD jails). Memory can also be depleted from within a jail.

Sun has borrowed from the FreeBSD jail concept to create zones in Solaris 10. Each zone shares the same operating system, similar to the FreeBSD jail. But Sun's implementation goes a bit further, by allowing separate resource allocation for each zone. Using resource management, CPU scheduling and memory usage can be limited for each non-global zone (every zone except the default, first zone is non-global and unable to see other zones). Sun has apparently solved the resource depletion problem found in the jail approach. I've heard that AIX and HP-UX use logical partitions to do something similar.

All of these approaches represent attempts to retrofit security on existing operating systems so that they can transparently support existing software. All the sysadmin has to do is leap through a few hoops, carefully and without making any egregious errors, and things will work. Hopefully.

Microsoft's current security model is so broken it deserves mention. There are way too many privileged processes running. And putting IE and its related HTML-rendering engine (used when reading email) on every system means that compromise is just an email message away. If ever an application screamed out to be put in a really effective jail, Internet Explorer is that application. Take IE, add default Administrator privileges for the first user account on every XP system, and IE becomes a root compromise, even when patched during Microsoft's (mostly) monthly patch announcement.

Now I invite you to follow along as I imagine a different world, a world where the operating system actually did prevent software, and even bad configuration, from creating root compromises. Let's start out with the type of system that Bill Cheswick alluded to in his invited talk, "My Dad's Computer." Cheswick's dad's computer had become like a lot of other Windows systems—so loaded with viruses, adware, spyware, and plain old cruft that it was barely usable. I've heard of other people buying a new Windows system and not connecting it to the Internet just so they could use a wordprocessor or spreadsheet without being interrupted by obscene popups every minute or so.

Most people need a simple desktop system that can do three things securely: play games, do Internet stuff, and handle simple office tasks. I mention playing games first because games have been driving the PC industries' quest for ever greater graphics performance, and because most of the people I see using laptops on airplanes are playing Solitaire. Games must be important. The fantasy OS must allow users to play games without affecting the overall state of the system (other than the DoS and heat discharge caused by pegging the CPU and graphics systems while displaying millions of 3-D polygons per second as sub-woofers shake the room).

The Internet stuff is much more of a problem. This is in which the operating system must maintain a lockbox where Web browsers, mail readers, and various IM tools can wreak all the havoc they want to without impacting the rest of the system. Of course, if users can continue to install software and plugins, even the lockbox will become so infested as to become unusable. So the user will get a button labeled "Clean up Internet" which cleans his own Internet lockbox, restoring it to a usable condition. Remember that I am waving a magic wand here, so I don't have to concern myself about cleaning up the cookies file, and preserving the state that the user actually cares about.

Some vendors like to use Web browsers as the mechanism for installing patches. If one can trust the operating system, signed patches can still be installed—by some service running outside the lockbox, after verifying signatures on the patches. Ideally, no patches would be required. But reality sometimes intrudes into even the wildest fantasies.

The final lockbox contains the user, her files, desktop, and office applications. These applications are sadly lacking in the ability to execute macros that might be included in documents or spreadsheets. The Internet lockbox can leave email and files in a directory where the user can access them. But no file stored anywhere in the user's file space can ever be executed.

If desktop systems were actually designed to function as business machines, things would be a lot simpler in this fantasy. The desktops could be diskless workstations where the user could never install any software, including viruses, spyware, adware, or games (sorry about that). Imagine centralized backup, identity management, software updates, and no more touching desktop systems. Sun's SunRay comes close to this.

Servers are headless. No fancy GUI software, nothing but command-line interfaces run using SSH. For the weak in sysadmin skills, fancy GUIs can be installed on desktop systems to issue the command lines used to configure and maintain the servers. No users except the administrator ever log in to the server. There are no accounts for them. The server does not include any software other than what is required for the operation and maintenance of installed services.

The server applications run in their own lockboxes, again preventing them from inadvertently damaging the systems they are hosted upon. The lockbox includes resource management controlling how the service behaves. For example, Web and SQL servers only accept connections; they never make outgoing connections. DNS servers send and receive packets on port 53. None of these services ever execs a shell. Services get allotted a reasonable fraction of total CPU, memory, and disk resources. While similar to a jail, this lockbox also includes resource allocation and limits on network activities. To make it easy to jump through configuration hoops, services come with configuration templates so that allocating resources is dead simple.

The fantasy OS would have to be tiny, so that it can be verified. We have seen more than enough kernel exploits in the last several years to convince people (at least me) that small is beautiful.

Frankly, the fantasy OS would permit us to get a lot more work done, as we would spend a lot less time dealing with broken systems.

I know I have left out a lot of necessary features, such as secure authentication that supports login and other services. You can read other people's ideas about single sign-on (Scher's article) and managing identification (Lear's article) in this issue. And just to balance things out, you can read about reverse engineering of code (Wysopal), slicker Windows rootkits (Butler and Sparks), defending against buffer overflows (Alexander), port knocking that unlocks SSH (Rash), and improvements to honeynets (Forte et al.). Jennifer Granick provides words of advice about the legality (or, rather, the lack thereof) of spyware. And Goel and Bush consider biological models for computer immune systems, because security will never be perfect.

Even in an imaginary world.

JAMES BUTLER AND SHERRI SPARKS

spyware and rootkits

THE FUTURE CONVERGENCE

Jamie Butler is the director of engineering at HBGary, Inc. and is co-author of the upcoming book Rootkits: The Day After. He is also a frequent speaker at computer security conferences, presenting his research on kernel reverse engineering, host-based intrusion prevention, and rootkits.

james.butler@hbgary.com

Sherri Sparks is a Ph.D. student of Computer Science at the University of Central Florida. Her current research interests are in software security, reverse engineering, and intrusion detection.

ssparks@longwood.cs.ucf.edu

ALL OF A SUDDEN YOUR PENTIUM 4, 3.2GHz desktop with 3GB of memory takes half an hour to boot into Windows. What's more, you can't seem to open Internet Explorer without being escorted to a home page you'd rather die than let your mother see. Of course, that is to say nothing of the unsolicited pop-up advertisements bombarding you at every click. And if all of that wasn't indication enough, you know there's a problem when your machine starts complaining about being "out of memory"...and the only program running is Notepad! Welcome, dear reader, to the modern worldwide scourge: spyware.

Understanding the Threat

WHAT IS SPYWARE?

The term "spyware" encompasses a large class of software capable of covertly monitoring a system and transmitting collected data to third parties. Such data may range from visited URLs to passwords and other confidential information. As of the 2003 publication of *Emerging Internet Threats Survey*,[1] one-third of companies have been affected by spyware-infected systems, and that number is growing. The implications are alarming on both commercial and private fronts. Of primary concern are the violations to personal privacy and the protection of intellectual property. Secondary issues relate to the degradation of system performance, network bandwidth, and utilization of IT personnel as they are forced to deal with application conflicts and system instability.

INFECTION VECTORS

A spyware infection is most often unintentional; however, there are cases where the software is purposely installed on someone's system to gather personal information or to monitor their Internet browsing habits. Commercial key loggers and parental control software both fall into this category. Unintentional infection may result from the exploitation of unpatched browser vulnerabilities and social engineering. It is not uncommon for unsuspecting users to be tricked into downloading and installing software they believe to be something else. Spyware may also piggy-back on the installation of another seemingly legitimate application. For instance, peer-to-peer networking software such as Kazaa and Bear Share, most of which contains embedded spyware, is proliferating. It goes without saying that the popularity of these programs ensures a continuing supply of infected hosts.

SPYWARE CHARACTERISTICS

We can divide spyware characteristics into primary and secondary traits, where primary traits relate to actively spying on the target system and secondary traits relate to concealing the spyware presence from the system's users. Primary traits include taking screenshots or logging keystrokes, running applications, and capturing URLs of visited Web pages. Sending logged information to a third party could also be considered a primary characteristic. In contrast, functionality for hiding registry entries, files, and running processes would be considered secondary traits. Behaviors that make the application difficult to remove also fall into this category. These types of behaviors include loading during the boot process, disabling detection programs, and reinstallation after removal.

Current-generation spyware is defined by a vast number of primary characteristics, but except for specific "hacker tools" (e.g., trojans), secondary characteristics have been either nonexistent or relatively primitive up to this point. In other words, most of this spyware is very efficient at spying but not very efficient at hiding. Consequently, it has, for the most part, been detectable with simple file- and registry-scanning techniques. Nonetheless, information security is a co-evolutionary process, and spyware development/detection is no exception. A few leading-edge spyware developers such as CoolWebSearch[2] are adapting and forcing us to an essential juncture. We note that the primary traits of spyware fundamentally describe a software trojan, while the secondary traits essentially define a rootkit. And we ask ourselves, What is the next generation in spyware? . . . Trojan meets Rootkit?

The Next Generation of Spyware: Trojan Meets Rootkit?

The functional requirements of a successful spyware application and a successful trojan or rootkit are remarkably similar. First, spyware programs, like rootkits and trojans, need to intercept user data such as keystrokes and network communications. Secondly, they must hide their presence from the user and/or make uninstallation difficult. Currently lacking sophistication in this second area, it is reasonable to expect that primitive spyware applications will continue to evolve their ability to conceal themselves. Rootkits have already mastered this ability. By understanding the application of advanced rootkit techniques to spyware, we may be better prepared to deal with the threat of an impending spyware epidemic.

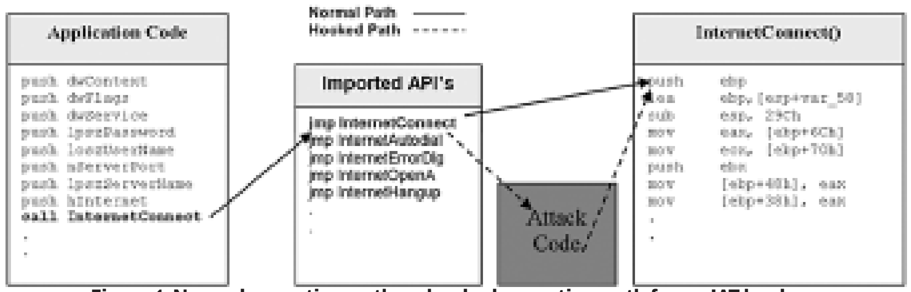
Regardless of the goal, whether it be hiding presence or intercepting communication, spyware and rootkits must both wedge themselves between the legitimate calling program, such as Internet Explorer, and the end communication point, either another node on the network or the underlying operating system. This involves altering normal program control flow. In order to accomplish this, rootkits use two primary technologies: hooks and layers. Once these technologies are in place, the spyware or rootkit can capture keystrokes as someone logs into her/his online bank account, or they can hide the presence of a particular process or network port from appearing on the local machine. In the remainder of this section we give an overview of rootkit techniques that are applicable to current and future spyware developments.

USER MODE TECHNIQUES (IMPORT HOOKING AND BROWSER HELPER OBJECTS)

Without a doubt, techniques such as import address table (IAT) hooking and browser helper objects (BHOs) are the most common methods of program subversion used by rootkits and trojans. Unfortunately, the Windows architecture makes these types of attacks accessible to even the lowest, most humble user mode applications. This is due to the fact that user programs and upper-level operating system components coexist at the same privilege level. Ultimately, spyware and rootkits have complete control over an application because they run with the same rights as the application they are hijacking. Some forms of spyware, such as key loggers and browser hijackers, already employ these techniques. Nevertheless, we expect to see an increase in their utilization as primitive spyware applications take on more rootkit-like characteristics to evade detection and removal.

In user mode, an attacker generally targets the APIs a program uses. This makes sense when you consider that user applications must rely upon the operating system to provide valuable functions such as opening files and writing to the registry. For example, if a spyware program is able to intercept a user mode scanner application's effort to open its files, it can return errors indicating those files don't exist. Subsequently, the scanner will falsely report that the system is uninfected. Windows APIs are implemented as dynamically linked libraries (DLLs) and are the basis of IAT hooking attacks.

The design of DLLs facilitates the attack. When an application uses an API function exported from a DLL such as InternetConnect in Wininet.dll, the compiler creates an `IMAGE_IMPORT_DESCRIPTOR` data structure in the application's binary file. The `IMAGE_IMPORT_DESCRIPTOR` contains the name of the DLL from which the function is exported and a pointer to a table containing all of the functions exported by the DLL that are used by the application. Each member of this table is an `IMAGE_THUNK_DATA` structure which is filled in at load time, by the Windows loader, with the memory address of the desired function. We can summarize the flow of execution as follows. Suppose an application makes a call to InternetConnect. First, the program code calls into the Import Address Table (IAT). From there, the IAT contains a jump that is taken to the destination address of the real function. It is easy to see that the IAT is a likely target for a rootkit or spyware. By changing a single function pointer, the attacker can re-route program execution through his/her own code, thereby capturing data, altering data, or even hiding the attacker's presence (see Figure 1). A more comprehensive explanation of the Windows portable executable (PE) format and IAT structure can be found in Matt Pietrek's "Inside Windows."[3]



```
push dwContext
push dwFlags
push dwService
push lpszPassword
push lpszUserName
push nServerPort
push lpszServerName
push hInternet
call InternetConnect
.
.
```

```
jmp InternetConnect
jmp InternetAutodial
jmp InternetErrorDlg
jmp InternetOpenA
jmp InternetHangup
.
.
```

```
push    ebp
lea     ebp, [esp+var_58]
sub     esp, 29Ch
mov     eax, [ebp+6Ch]
mov     ecx, [ebp+70h]
push    ebx
mov     [ebp+40h], eax
mov     [ebp+38h], ecx
.
.
```

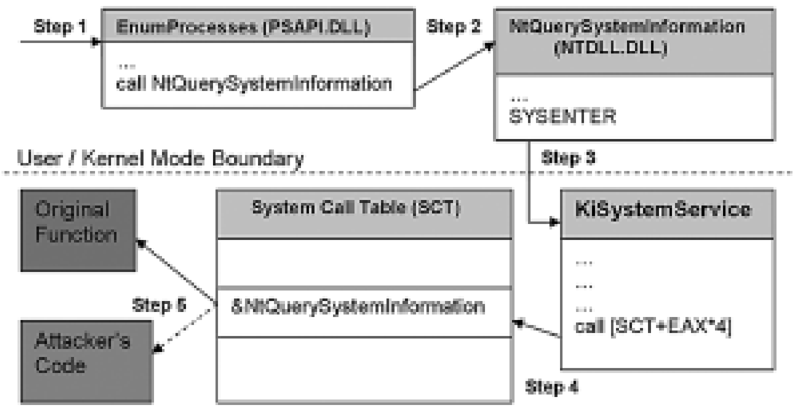
Figure 1. Normal execution path vs. hooked execution path for an IAT hook

Browser Helper Objects (BHOs) demonstrate another DLL-based userland attack technique. While BHOs are specifically designed to customize and extend Internet Explorer, many browsers provide similar features that can be maliciously exploited by spyware applications. A BHO is suspect if a user suddenly notices that his home page has been redirected, "new" toolbars have suddenly appeared in his browser, or his list of "favorites" has been modified. The risk of BHOs, however, extends beyond the mere inconvenience of having one's homepage hijacked. By definition, BHOs are in-process Component Object Model (COM) DLLs which Internet Explorer loads on startup. The result is that a BHO has complete access to Explorer's process address space. In practical terms, this means a BHO can intercept all of the events occurring in the user's browser. For example, the `BEFORENAVIGATE2` event is triggered before Internet Explorer navigates to a Web page. This means that a BHO has access to the URL before the page is even downloaded. More alarming is the fact that BHOs are not limited to acting on browser events. Really, anything is possible within the constraints of the permissions of the user who launched Internet Explorer. This includes creating or deleting files, executing programs, reading email, and recording and sending private Internet banking information. What makes BHOs particularly troubling is that it is not obvious that they are running. Since they run as a DLL within Internet Explorer, it is almost impossible to distinguish a malicious BHO from a completely benign one.

KERNEL MODE TECHNIQUES (CALL TABLE HOOKING AND FILTER DRIVERS)

In our discussion of user mode we noted that user applications coexist with some portions of the operating system. We now extend that statement to kernel mode. At this highest of privilege levels, drivers coexist with the Windows kernel itself. This means that a malicious driver has the power to usurp complete control of the operating system environment. Indeed, modern rootkits have reached an alarming degree of sophistication in their employment of kernel mode hooking techniques. Fortunately, writing a kernel driver is something of a "black art," so that, with the exception of a very few advanced key loggers, most spyware developers haven't caught up to the rootkit developers *yet*. Nevertheless, as spyware continues to evolve in complexity and stealth, these techniques may become a very real threat to information security. In the following discussion we cover three of the most common kernel hooking techniques: system call table hooking, filter drivers, and IRP table hooking.

The system call table (SCT) is one of the simplest, most effective places to hook in the kernel. It provides the gateway into the kernel through which all user mode API calls must pass. In most operating systems the SCT is implemented as a table of pointers to the functions the kernel exports to user mode applications. The Windows system call mechanism is also based on this concept. Calls to the Kernel32.dll and Ntdll.dll API functions pass through the kernel function called KiSystemService, which does some sanity checking on the function parameters and then calls the referenced SCT function. By modifying the function pointer in the table to point to attack code, an attacker has total control over the operation of the function (see Figure 2). Spyware and rootkits can use this trick to filter information they do not want the user or system administrator to see. For example, by hooking NtQuerySystemInformation in the SCT and filtering its response, the attacker can hide any file or directory in Windows. Although the idea of modifying a table of function pointers is reminiscent of userland IAT hooking, kernel SCT hooking is a much more powerful technique. Where IAT



Step 1
EnumProcesses (PSAPI.DLL)
...
call NtQuerySystemInformation
Step 2
NtQuerySystemInformation (NTDLL.DLL)
...
SYSENTER
User / Kernel Mode Boundary
Step 3
Original Function
System Call Table (SCT)
KiSystemService
Step 5
&NtQuerySystemInformation
...
...
...
call [SCT+EAX*4]
Attacker's Code
Step 4

Figure 2. Normal execution path vs. hooked execution path for an SCT hook

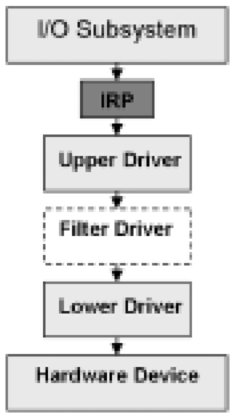


Figure 3. Windows has a layered driver architecture

hooking is local to the application process being hooked, an SCT hook will globally intercept functions across all processes, including the operating system itself.

In Windows, the drivers for a system's hardware devices are layered into a hierarchal "device stack." Furthermore, a given hardware device may have one or more drivers associated with it, which we can visualize as a vertical stack of layered components (see Figure 3). These drivers communicate with each other and the operating system by means of I/O request packets (IRPs). Filter drivers and IRP hooking techniques exploit the layered nature of Windows' driver architecture. Unlike normal drivers, filter drivers are transparently inserted on top of or in between existing drivers in the stack, using the kernel API, IoAttachDevice. Although they are sometimes legitimately used to add functionality to an existing lower-level driver, an attacker will typically use them to either modify or intercept data. Key loggers and network sniffers typically use them to capture user passwords and other sensitive information.

By design, layered filter drivers require a lot of code to implement. Furthermore, they may be more easily detected than a driver that does direct IRP hooking. Instead of installing a filter, an attacker can directly hook the functions exported by a target device driver in its IRP major function table. The IRP major function table is simply an array of 28 function pointers to handler functions in response to notifications and requests which the driver receives from either a client application or the operating system. An application typically sends an IRP to a driver to request a specific service. For example, `IRP_MJ_DIRECTORY_CONTROL` is sent to file system drivers to request the list of directories and files. By intercepting and altering different I/O requests, spyware can easily hide on the file system or eavesdrop on network communications. Some commercial software such as ZoneAlarm[4] even uses this technique to intercept and regulate network traffic. It is not difficult for an attacker to find a particular driver object in memory. Windows' kernel provides the function called IoGetDeviceObjectPointer which spyware or a rootkit can call to get a pointer to the named device object. This device object contains a pointer to its corresponding driver object which the attacker can use to reference the target's IRP function table. Figure 4 shows the relationship among these objects in memory.

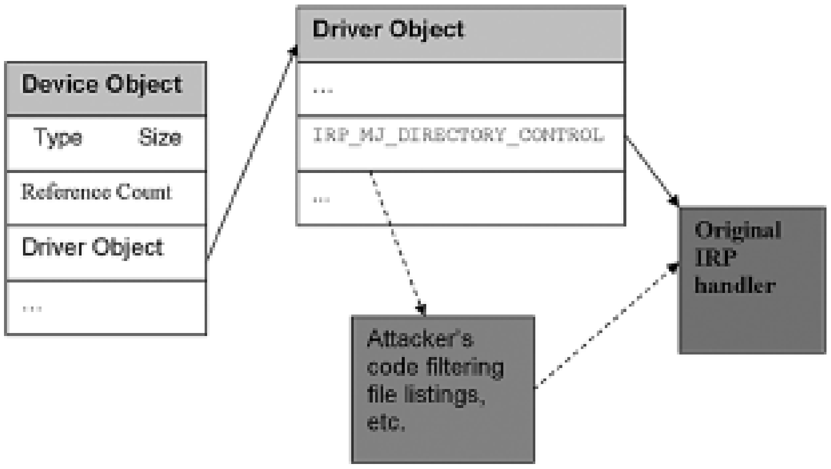


Figure 4. Illustration of hooking a driver's IRP table

HYBRID TECHNIQUES (INLINE FUNCTION HOOKING)

Inline function hooking can be considered a hybrid technique, since it can be applied either to a user mode application or to the kernel. This technique is a bit more advanced and harder to detect than the methods previously mentioned. Rather than simply replacing a pointer in a table, an inline hook alters the target function itself. Normally this is done by replacing the first few bytes of the function with an unconditional jump to the rootkit or spyware code. Before this overwrite occurs, the attacker saves the bytes being replaced, so the semantics of the original function are maintained. The trick here is in the fact that instructions are variable length on an X86 processor. Therefore, although an unconditional jump is only five bytes on a 32-bit architecture, the instructions being overwritten may have a different length. Inline function hooking becomes extremely difficult to detect if the jump is embedded deeply in the target function. Complicating matters further, the destination of the jump may be nondeterministic except at the moment of execution. A clever piece of spyware has the full breadth of the assembly language and all of its potential permutations within which to hide.

Managing the Threat: Spyware Detection

There are a number of commercial and freeware spyware detection tools available. Like most things free, some of them are better than others. We found BHO Demon very useful in detecting and disabling BHOs in Internet Explorer.[5] It installs a service on the local machine to watch for future attempts to inject BHOs into Internet Explorer. Spybot Search & Destroy was also very useful during our research.[6] It not only detects BHOs, but additionally detects and removes other forms of spyware and adware.

SPYWARE DETECTION CHALLENGES

Current spyware detection tools are primarily based upon signature scanning techniques. Signature scans have been used heavily by antivirus (AV) engines and are quite reliable for detecting known strains of spyware. Unfortunately, they are ineffective against unknown strains, which must first be caught, analyzed, and sampled for a usable signature. With new spyware variants emerging almost daily, it is difficult for detection engines to keep pace. Indeed, there are even a handful of spyware applications which utilize a rudimentary form of polymorphism to randomize their file names and registry keys, so that every infected machine contains a slightly different version of the program. This makes it more difficult for a detection program to obtain a consistent signature for the application. Some detectors have turned to heuristics to deal with these issues. A further problem with current detections lies in the fact that many of them run in user mode right alongside the spyware applications they are attempting to apprehend. Using the aforementioned hooking techniques, a malicious spyware application is capable of intercepting the function calls of a user mode detection engine as easily as it hijacks the user's Internet browser. In this manner, clever spyware may trick the detector into believing the machine is uninfected. A detection engine implemented in kernel mode will provide defense against this attack as long as spyware remains a mostly user mode phenomenon.

VICE

VICE is a freeware tool designed to detect hooks.[7] It is based upon heuristic analysis of hooking behaviors rather than exact signatures. The benefit of this approach is that VICE is capable of pinpointing suspicious activity related to previously unknown rootkits or spyware. It is implemented as a stand-alone program capable of analyzing both user mode applications and the operating system. VICE checks the address space of every application looking for IAT hooks in every DLL that those applications use. It also checks the kernel SCT for function pointers that do not resolve to ntoskrnl.exe and the IRP major function tables for a list of user defined drivers. A user can add devices to this list by editing the driver.ini file. Inline function hooks are detected in DLL functions imported by applications, as well as in the kernel SCT functions themselves. When possible, VICE will display the full path on the file system of the DLL or device driver doing the hooking, so that a system administrator can examine and remove the malicious software. It should be noted, however, that VICE is not an end-user spyware detection and removal tool. Some legitimate applications such as firewalls and antivirus products also use these techniques to filter and examine data, so an operator of VICE will need to have experience enough to recognize those cases. As stated previously, many current spyware applications are immature and do not utilize advanced hooking techniques. Nevertheless, as spyware evolves, VICE stands to become an increasingly useful tool, as it has proved to be in the battle against Windows rootkits. Today, VICE will detect most publicly known Windows rootkits and any spyware that currently uses these more intrusive hooking technologies. To run VICE, the host machine must have the Microsoft .NET Framework installed, which is free for download.

Conclusion

We hope to advance VICE as rootkit and spyware techniques continue to evolve. The sophistication of the disassembly engine logic can be improved to identify more complex and/or deeply embedded inline function hooks. In future versions, VICE will become more of an active forensics tool, with enhanced capabilities for detecting anomalous registry accesses, file accesses, and network communications on systems suspected of being compromised. Although it is still an open-ended topic of research, host-based systems tend to have a better understanding of their environment than do Network Intrusion Detection Systems (NIDS). This translates into an advantage for heuristic systems like VICE that rely on their ability to differentiate between "normal" and "abnormal" behaviors.

Spyware has become a threat to corporate and personal information security. With the combined goals of data interception and stealth, a spyware application is well suited to leverage both trojan and rootkit technologies. Although current spyware lacks the sophistication of modern rootkits for hiding its presence on a system, we can expect that to change with the advent of more advanced detection and removal tools. In the next generation of spyware we expect to see more complex hooking, polymorphic techniques, and kernel mode components. By understanding the potential application of rootkit stealth techniques to spyware, hopefully we will be better prepared to meet the coming challenges in detection and removal.

REFERENCES

1. Websense, "Emerging Internet Threats Survey 2003": http://www.websense.com/company/news/research/Emerging_Threats_2003_EMEA.pdf.

2. CoolWebSearch: http://www.coolwebsearch.com.

3. Matt Pietrek, "Inside Windows: An In-Depth Look into the Win32 Portable Executable File Format." *MSDN Magazine*, vol. 17, no. 2 (February 2002).

4. ZoneAlarm Pro: http://www.zonelabs.com.

5. BHO Demon: http://www.definitivesolutions.com/.

6. Spybot Search & Destroy, by Patrick M. Kolla: http://www.safer-networking.org/.

7. VICE, by James Butler, HBGary Inc.: http://www.rootkit.com/vault/fuzen_op/vice.zip.

CHRIS WYSOPAL

putting trust in software code

Chris Wysopal is director of development at Symantec Corporation, where he leads research on how to build and test software for security vulnerabilities.

chris_wysopal@symantec.com

TWENTY YEARS AGO, KEN THOMPSON, the co-father of UNIX, wrote a paper about the quandary of not being able to trust code that you didn't create yourself. The paper, "Reflections on Trusting Trust,"[1] details a novel approach to attacking a system. Thompson inserts a back door into the UNIX login program when it is compiled and shows how the compiler can do this in a way that can't be detected by auditing the compiler source code. He writes:

"You can't trust code that you did not totally create yourself. No amount of source-level verification or scrutiny will protect you from using untrusted code. In demonstrating the possibility of this kind of attack, I picked on the C compiler. I could have picked on any program-handling program such as an assembler, a loader, or even hardware microcode."

Twenty years after Thompson's seminal paper was published, developments in the field of automated binary analysis of executable code are tackling the problem of trusting code you didn't write. Binary analysis can take on a range of techniques, from building call trees and looking for external function calls to full decompilation and modeling of a program's control flow and data flow. The latter, which I call deep binary analysis, works by reading the executable machine code and building a language-neutral representation of the program's behavior.

This model can be traversed by automated scans to find security vulnerabilities caused by coding errors and to find many simple back doors. A source code emitter can then take the model and generate a human-readable source code representation of the program's behavior. This enables manual code auditing for design-level security issues and subtle back doors that will typically escape automated scans.

The steps of the decompilation process are as follows:

1. Front end decodes binary to intermediate language.
2. Data flow transformer reconstructs variable lifetimes and type information.
3. Control flow transformer reconstructs loops, conditionals, and exceptions.
4. Back end performs language-specific transformation and exports high-level code.

To be useful the model must have a query engine that can answer questions for security scanning scripts:

- What is the range of a variable?
- Under what conditions is some code reachable? Any path, all paths?
- What dangerous actions does this program perform?

Security scanning scripts can then ask questions such as:

- Can the source string buffer size of a particular unbounded string copy be larger than the destination buffer size?
- Was the return value from a security-critical function call tested for success before acting on the results of the function call?
- Is untrusted user input used to create the file name passed to a create-file function?

By building meaning from the individual instructions that are executed by the CPU, deep binary analysis understands program behavior that is inserted by the compiler. Thompson's back-door code can't hide from the CPU, and it can't hide from deep binary analysis. More important for real-world software security is that vulnerabilities and back doors can't hide in the static and dynamic libraries that a program links to and for which source code is not always available.

The programmer productivity benefits of using off-the-shelf software components are well known, but not much is said about the risks of using binary components, which are common on closed source operating systems. When developing enterprise applications, programmers frequently concentrate on writing business logic and leave presentation, parsing, transaction processing, encryption, and much more to commercial libraries for which they often have no source code. These libraries typically come from OS vendors, database vendors, transaction-processing vendors, and development framework vendors. Often the programmers dutifully audit the 20% of the application they wrote and ignore the 80% they cannot audit. Yet it is the entire program that is exposing the organization running it to risk.

Deep binary analysis can follow the data flows between the main program and the libraries in use to find issues that arise from the interaction between the main program and a library function. Often any security vulnerabilities discovered can be worked around by putting additional constraints on the data passed to the library function. Sometimes, however, the function call will need to be replaced. Using the tools we have developed at @stake (now Symantec), we have found buffer overflows in the string functions of a popular C++ class library and poor randomness being used in a cryptolibrary function. The cryptolibrary function was using a random number generated by `rand()`, and `srand()` was seeded with zero. No doubt there was a comment in the C code stating that the random number generation needed to be replaced in the future, but this being binary analysis we couldn't tell.

An interesting use of binary analysis is the differential analysis of two binaries that have small differences between them. Perhaps you are engaged in incident response and have discovered an altered binary on the system for which the attacker has not left any source code behind. Differential binary analysis can be used to see what behavior has been added or removed from the binary. A use that has important security implications is to reverse engineer the details of a vulnerability by determining the differences between a vulnerable program and one that has a vendor security patch applied. If black hats can easily determine the cause of a vulnerability simply by looking at the patch, then there is little to be gained from vendors withholding vulnerability details, and there is increased urgency to patch vulnerable systems quickly.

Halvar Flake has developed tools for binary diffing and gave a presentation on his techniques at Black Hat Windows 2004.[2] Todd Sabin has developed different techniques for differential binary analysis which he calls "Comparing Binaries with Graph Isomorphisms."[3] The field is rapidly evolving, so as with so many topics in the security arena, defenders are urged to stay apprised of developments, because attackers surely will.

I am hopeful that the field of binary analysis will evolve to a point where third-party testing labs will be able to perform repeatable, consistent security testing on closed source software products without the cooperation of software vendors. Much as *Consumer Reports* is able to verify automobile vendor claims of performance and carry out their own safety testing, a software testing lab would be able to quantify the number of buffer overflows, race conditions, script injections, and other issues in a program under automated test. A security quality score could be generated from the raw test results. It will undoubtedly be imperfect—there will always be issues missed and some false positives—but on a coarse scale, say, ranking program security from A to F, it would be very useful to consumers.

Today there is little useful information by which to rate software security quality except for a particular product's security patch record. This record is somewhat useful but typically only exists for the most popular products, which garner the bulk of the attention of security researchers. Another source of security information is common criteria evaluations. But with products that have received common criteria EAL 4 certification still being subject to monthly patches of critical severity, there is clearly a need for additional ways of rating security quality.

Deep binary analysis stands to revolutionize the software security space not only for developers and businesses but for consumers, too. It's an exciting future, one in which we don't have to trust the compiler manufacturers, third-party driver and library providers, or application and operating system vendors. Software developers can use binary analysis tools to discover and remediate the vulnerabilities in their own software, and consumers can verify that their vendors have performed due diligence and are not delivering shoddy code.

REFERENCES

1. Ken Thompson, "Reflections on Trusting Trust," Communications of the ACM, vol. 27, no. 8 (August 1984), reprinted at http://www.acm.org/classics/sep95.

2. Halvar Flake, "Automated Binary Reverse Engineering": http://www.blackhat.com/presentations/win-usa-04/bh-win-04-flake.pdf.

3. Todd Sabin, "Comparing Binaries with Graph Isomorphisms": http://www.bindview.com/Support/RAZOR/Papers/2004/comparing_binaries.cfm.

MICHAEL RASH

combining port knocking and passive OS fingerprinting with fwknop

Michael Rash holds a master's degree in applied mathematics and works as a security research engineer for Enterasys Networks, Inc. He is the lead developer of the cipherdyne.org suite of open source security tools, including PSAD and FWSnort, and is co-author of the book Snort-2.1 Intrusion Detection published by Syngress.

mbr@cipherdyne.org

IT WAS AROUND 2:45 A.M. ONE SUMMER night in 2002 and I had finally finished. My shiny new Linux system was fully installed and connected to my broadband cable modem connection in my apartment. All unnecessary services had been turned off, a restrictive iptables policy had been deployed, and a tripwire filesystem snapshot had been taken, all before connecting the system to the network.

Reasoning that the only servers I needed to have accessible from arbitrary IP addresses around the Net were Apache and OpenSSH, iptables could be configured to log and drop almost all connection attempts. After connecting the system to the network and scanning it from a shell account on a different external network, I saw that only TCP ports 22, 80, and 443 were accessible, so I was satisfied that my system was fit to remain connected.

It was late, though, and I forgot one important detail. I neglected to check the version of OpenSSH that came bundled with the Linux distribution. Back then RedHat 7.3 was my Linux distribution of choice even though more recent versions of RedHat (and other Linux distributions) were available. After getting some well-deserved sleep, I woke the next morning only to find that, sure enough, my system had been compromised. Luckily, I had no important data on the box yet, but it could have been a lot worse.

It became clear that in addition to upgrading to the latest version of sshd, it would also be desirable to protect sshd as much as possible with iptables. Yet at the same time, the ability to log in remotely and administer the system from anywhere was highly desirable. Unfortunately, these two goals are fundamentally at odds. Sure, sshd does not allow just anyone to log in or execute commands; users must have the proper authentication credentials for at least one type of authentication method supported by sshd (username/password, RSA, Kerberos, etc.), but all of this may not help if there is a buffer overflow vulnerability (as in my case) buried within a section of the sshd code that is accessible over the network.

An attacker may only need the capability of connecting to sshd in order to be in a position to exploit such a vulnerability. Being able to connect means the attacker can send packets up through the server's IP stack, establish a TCP session with the transport layer, and, finally, talk directly to sshd. Alternatively, if iptables does not allow the attacker's IP to connect to sshd, then any packets sent from the attacker are blocked by iptables before they even make it into the IP stack in the Linux kernel, let alone to the SSH daemon itself.

Clearly, the most desirable way to protect an arbitrary service is with iptables. However, since not all IP addresses that should be allowed to connect to sshd can be enumerated a priori, I would need to add an additional authentication layer to iptables. Port knocking provides a simple but effective solution to this problem.

Port Knocking

Port knocking[1] provides a method of encoding information within sequences of connection attempts to closed (or open) ports. The most common application of such information (which can include IP addresses, port numbers, protocols, usernames, etc.) is the modification of firewall policies or router ACLs in response to monitored valid port knock sequences. In essence, port knocking provides a means of network authentication that only requires the ability to send packets that contain transport layer headers.

Knock clients do not need to actually talk to a server-side application or even have a TCP session established; the knock server can behave completely passively as far as network traffic is concerned. On the server side, knock sequences can either be monitored via firewall log messages or with a packet capture library such as libpcap, in the same way an IDS watches traffic on a network. Although using a packet capture library would provide the ability to encode information such as a password at the application layer, a full-featured port knock implementation is completely feasible without using any packet data above the transport layer—hence firewall logs are ideally suited for this application. Implementing a port knocking scheme around firewall logs has the added bonus of helping to ensure the firewall is configured correctly, or at least that it is logging packets.

Protecting Against Replay Attacks

It should be noted that many port knocking techniques are susceptible to replay attacks if an attacker is in a position privileged enough to be able to sniff traffic between the port knock client and server. An attacker need only replicate a knock sequence for the server to grant the same level of access that would be granted to a legitimate client. Hence some would argue that port knocking suffers from the standard arguments against "security through obscurity." However, port knocking is not designed to act as the only security mechanism for secure communications; encryption implemented by sshd serves as the main line of defense. Port knocking provides an additional layer of security on top of the secure communications already implemented by sshd. The argument against "security through obscurity" is only valid if security is *completely* dependent on obscurity.[2] In addition, there are several techniques for raising the bar for the attacker even if the entire sequence has been observed on the wire. Let us examine four such techniques:

1. Relative timings between sequence packets can be made significant. For example, the knock server may require that the minimum time delay between successive knock sequence packets is at least three seconds, but not longer than six seconds.
2. Multiple protocols (TCP, UDP, and ICMP3) can be used within the knock sequence. If an attacker has restricted the view of a sniffer to just, say, the TCP protocol, then some portion of such a sequence will be missed and hence cannot be replayed on the network.

3. Encryption can be used. Due to the fact that an IP address, a protocol number, and a port number together only require seven bytes of information to represent, it is easy to use a symmetric block cipher (such as the Rijndael algorithm) to encrypt port knock sequences before they are sent across a network. Encrypting an IP address within a knock sequence allows a knock client to instruct a knock server to allow access for a third-party IP address that cannot be guessed by anyone observing the knock sequence. As usual, use of a symmetric encryption algorithm requires a shared key that is known to both the knock client and the knock server. There are also fancier methods of using encryption, such as one-time passwords,4 that genuinely make replay attacks infeasible.
4. Requirements can be made on the type of operating system that generates a knock sequence. Additional fields in the IP and TCP headers—TTL values, fragment bits, overall packet size, TCP options, TCP window size, etc.—can be made significant. If a knock sequence is monitored between a client and server, then any duplicated sequence will not be honored by the server unless the OS of the duplicate sequence exactly matches that of the original client. For example, if a knock sequence between two Linux machines is sniffed off the wire and an attacker replays the sequence from a MacOS X machine, the duplicate sequence will be ignored. Of course, OS characteristics can be spoofed by the attacker, but this may not be worth the trouble (again, although this is not unbreakable, port knocking adds an additional layer of security).

Fwknop

This article discusses a tool called fwknop (Firewall Knock Operator), which supports both shared and encrypted port knock sequences along with all four of the obfuscation techniques mentioned above. fwknop exclusively uses iptables log messages to monitor both shared and encrypted knock sequences instead of appealing to a packet capture library. In addition, due to the completeness of the iptables logging format, fwknop is able to passively fingerprint operating systems from which connection attempts originate. fwknop uses this capability to add an additional layer of security on top of the standard knock sequences by requiring that the TCP stack that generates a knock sequence conform to a specific OS. This makes it possible to allow, say, only Linux systems to issue a valid knock sequence against the fwknop knock server. I develop and release fwknop as free and open source software under the GNU Public License (GPL); fwknop can be downloaded from http://www.cipherdyne.org/fwknop/.

IMPLEMENTATION

Firewall logs, especially those created by iptables, can provide a wealth of information about port scans, connection attempts to back door, DDoS programs, and attempts by automated worms to establish connections to vulnerable software. One of the most important characteristics of firewall logs is that packets can be logged completely passively; the firewall is under no obligation to allow the target TCP/IP stack to generate any return traffic in response to a TCP connection attempt. Yet, at the same time, all sorts of juicy bits of information can be logged from a connection attempt, such as TTL and IP ID values, source and destination port numbers, TCP flags, TCP options, and more. (Note that UDP and ICMP packets will generate iptables log messages that contain information appropriate to those protocols.)

fwknop parses iptables log messages that are sent to syslog as iptables intercepts packets that traverse the firewall interfaces. By default, iptables logs packets via the syslog `kern` facility at a priority of `info`. Such messages are usually sent to the file /var/log/messages, but fwknop reconfigures syslog to also send kern.info messages to a named pipe, where they are read by fwknop. Let us examine an iptables log message generated by the following iptables rule:

```
iptables -A INPUT -p tcp -i eth0 -j LOG —log-tcp-options
```

A TCP syn packet to port 60000 on the eth0 interface will result in the following log message logged via syslog to /var/log/messages:

```
Aug  7 17:22:57 orthanc kernel: IN=eth0 OUT=
MAC=00:0c:41:24:68:ef:00:0c:41:24:56:37:08:00 SRC=192.168.10.2 DST=10.3.2.1 LEN=60
TOS=0x10 PREC=0x00 TTL=64 ID=56686 DF PROTO=TCP SPT=32811 DPT=60000 WINDOW=5840 RES=0x00
SYN URGP=0 OPT (020405B40402080A06551B7A0000000001030300)
```

Iptables does a good job of decoding packet information before sending it to syslog. Clearly displayed (among other things) are source and destination IP addresses, packet length and TTL values, source and destination ports, TCP window size, and TCP flags. The TCP options portion of the TCP header is also visible, but because decoding it would place an undue burden on the kernel, only the raw options data is logged. In an effort to passively fingerprint the operating system that generated the above log message, fwknop uses a strategy similar to p0f,[5] which is one of the best passive OS fingerprinters available. Since matching a p0f signature against the packet above requires the examination of specific TCP option values, fwknop must decode the options string. A quick examination of RFC 793 informs us that there are two formats for TCP options: three 8-bit-wide fields denoting the option type, length, and value, or a single 8-bit-wide field denoting the option type. Interpreting these two formats along with the appropriate TCP option definitions with an eye toward what is required by p0f, fwknop decodes the options string in the packet above,

`020405B40402080A06551B7A0000000001030300`,

as the following:

```
- Maximum segment size = 5840  - Selective acknowledgment is permitted  - The timestamp
is set  - NOP  - Window scale = 0
```

Hence, the packet log message above is matched by the following p0f signature:

```
S4:64:1:60:M*,S,T,N,W0     Linux:2.4::Linux 2.4/2.6
```

Now let's turn to some concrete port knocking examples. The following two knock sequence examples will involve the execution of fwknop from the command line in client mode from the source IP 192.168.10.2 to the destination machine 10.3.2.1, where fwknop is running in server mode. (RFC 1918 addresses were chosen for illustration purposes so as not to step on the toes of any real networks out there.) In both sequence examples iptables is configured to block access to sshd on the knock server, but after receiving a valid port knock sequence, fwknop will reconfigure iptables to allow access to sshd. In order to indicate clearly how access is modified, connection attempts to sshd on the knock server will be made from the knock client system before and after sending the knock sequences. As fwknop receives and parses knock sequences and modifies access controls, it writes information to syslog, and these messages will also be displayed below. All command-line invocations of fwknop below take place on the client system.

SHARED SEQUENCE

First let's examine a shared sequence that involves multiple protocols. Fwknop supports the use of TCP, UDP, and ICMP echo requests within shared knock sequences. Shared sequences must be defined in two places: the file ~/.fwknoprc on the client system, and the file /etc/fwknop/access.conf on the server system. Hence, our first knock sequence is defined as follows on the server:

```
[server]# cat /etc/fwknop/access.conf
SOURCE: ANY;
SHARED_SEQUENCE: tcp/50053, udp/6020, icmp, icmp,
tcp/24034, udp/9680;
OPEN_PORTS: tcp/22;
FW_ACCESS_TIMEOUT: 30;
REQUIRE_OS_REGEX: linux;
```

The SOURCE keyword defines from which IP address or network a knock sequence will be accepted (with the special value ANY accepting knock sequences from any source IP). The SHARED_SEQUENCE keyword defines the specific port numbers and protocols that constitute a valid sequence. The OPEN_PORTS keyword defines the set of ports and corresponding protocols to which the source address should be allowed to connect. Fwknop will reconfigure iptables on the underlying Linux system only upon receiving a valid knock sequence. The FW_ACCESS_TIMEOUT specifies the length of time (in seconds) the underlying iptables policy will be configured to accept connections from an IP address that has issued a valid knock sequence. The REQUIRE_OS_REGEX variable instructs fwknop to accept a knock sequence if and only if the p0f signature derived from the originating operating system contains the specified string (the match is performed case-insensitively).

In the file ~/.fwknoprc on the client system, a similar block of text defines the same sequence for the fwknop client. Note the specific IP address of the fwknop server is listed immediately preceding the sequence definition:

```
[client]$ cat ~/.fwknoprc
10.3.2.1: tcp/50053, udp/6020, icmp, icmp, tcp/24034,
udp/9680
```

Now for the actual execution of fwknop. First, connectivity to sshd is tested from the client, then the port knock sequence is sent across the network to the server, and, finally, an additional connection attempt shows that access has indeed been granted:

```
[client]$ telnet 10.3.2.1 22
Trying 10.3.2.1...
[client]$ fwknop -k 10.3.2.1
[+] Sending port knocking sequence to knock server:
10.3.2.1
[+] tcp/50053 -> 10.3.2.1
[+] udp/6020  -> 10.3.2.1
[+] icmp echo request -> 10.3.2.1
[+] icmp echo request -> 10.3.2.1
[+] tcp/24034 -> 10.3.2.1
[+] udp/9680  -> 10.3.2.1
[+] Finished knock sequence.
[client]$ telnet 10.3.2.1 22
Trying 10.3.2.1...
Connected to 10.3.2.1.
Escape character is ‘^]’.
SSH-2.0-OpenSSH_3.8.1p1
```

On the server the following messages are written to syslog by fwknop as it monitors the port knock sequence in the iptables log:

```
Aug 8 13:17:46 orthanc fwknop: port knock access sequence matched for 192.168.10.2
Aug  8 13:17:46 orthanc fwknop: OS guess: Linux:2.4::Linux 2.4/2.6 matched for
192.168.10.2
Aug  8 13:17:46 orthanc fwknop: adding INPUT ACCEPT rule for source: 192.168.10.2 to con-
nect to tcp/22
Aug  8 13:18:18 orthanc fwknop: removed iptables INPUT ACCEPT rule for 192.168.10.2 to
tcp/22, 30 second timeout exceeded
```

The log shows that the fwknop server added a rule in the iptables INPUT chain for a total of 30 seconds to accept connections from 192.168.10.2 over tcp/22. Although the 30-second timeout seems a bit short, if the iptables policy on the underlying system is written so that packets that are part of established sessions are accepted first before remaining packets are dropped, then any SSH session that was established within the 30-second window will not be killed when the ACCEPT rule is removed. Note that the port number for which the fwknop server permitted access never appears in the knock sequence itself; it is defined in /etc/fwknop/access.conf, so the client has to know to which port(s) it has access after sending the sequence. This characteristic holds true for all shared sequences.

ENCRYPTED SEQUENCE

Now let's take a look at an encrypted knock sequence. This time the sequence itself will change depending on the key used to encrypt the source IP address, protocol, port number, and local username that fwknop is being executed as. Thus, encrypted sequences are not defined within any configuration file on the server or client systems. Sequences are monitored on the server; if successfully decrypted then such a sequence is valid and access controls will be modified. The fwknop server must still be configured with the appropriate encryption key and port(s) to open, and as usual this information is contained in the /etc/fwknop/access.conf file on the fwknop server:

```
[server]# cat /etc/fwknop/access.conf
SOURCE: ANY;
ENCRYPT_SEQUENCE;
KEY: 3ncryptk3y;
OPEN_PORTS: tcp/22;
FW_ACCESS_TIMEOUT: 30;
REQUIRE_OS_REGEX: linux;
```

The SOURCE, OPEN_PORTS, FW_ACCESS_TIMEOUT, and REQUIRE_OS_REGEX keywords are used as before, but two additional keywords, ENCRYPT_SEQUENCE and KEY, are defined to instruct fwknop to accept a port knock sequence encrypted with the subsequent key. Now for our encrypted port knocking example:

```
[client]$ telnet 10.3.2.1 22
Trying 10.3.2.1 . . .
[client]$ fwknop -e -a 192.168.10.2 -P tcp -p 22 -r -k
10.3.2.1
[+] Enter an encryption key (must be as least 8 chars, but
less than 16
chars). This key must match the key in the file
/etc/fwknop/access.conf
on the remote system.
[+] Encryption Key:
[+] clear text sequence: 192 168 10 2 0 22 6 28 109 98 114
0 0 0 0
[+] cipher text sequence: 182 246 253 35 195 76 157 229 86
13 152 30 120 172 58 140
[+] Sending port knocking sequence to knock server:
10.3.2.1
[+] tcp/61182 -> 10.3.2.1
[+] udp/61246 -> 10.3.2.1
[+] tcp/61253 -> 10.3.2.1
[+] udp/61035 -> 10.3.2.1
[+] tcp/61195 -> 10.3.2.1
 . . .
```

```
[+] Finished knock sequence.
[client]$ telnet 10.3.2.1 22
Trying 10.3.2.1 . . .
Connected to 10.3.2.1.
Escape character is ‘^]’.
SSH-2.0-OpenSSH_3.8.1p1
```

On the server the following messages are written to syslog by fwknop:

```
Aug  8 13:00:28 orthanc fwknop: decrypting knock sequence for 192.168.10.2
Aug  8 13:00:28 orthanc fwknop: OS guess: Linux:2.4::Linux 2.4/2.6 matched for
192.168.10.2
Aug  8 13:00:28 orthanc fwknop: username mbr match
Aug  8 13:00:28 orthanc fwknop: adding INPUT ACCEPT rule for source: 192.168.10.2 to con-
nect to tcp/22
Aug  8 13:01:00 orthanc fwknop: removed iptables INPUT ACCEPT rule for 192.168.10.2 to
tcp/22, 30 second timeout exceeded
```

Note that fwknop has allowed the real source through the firewall since this is the source address that was encrypted in the sequence with the `-a` option. Any other third-party IP address could have been specified here. Also notice that as with the previous shared sequence, fwknop passively fingerprinted the client operating system and the required Linux OS was found. Finally, note that the encrypted sequence is rotated through the TCP and UDP protocols. More information about fwknop, including a detailed description of all configuration directives, can be found at http://www.cipherdyne.org/fwknop/.

Conclusion

Port knocking adds an additional layer of security for arbitrary services that are accessible over a network. A client system must send a specific sequence of connection attempts to the knock server before access is granted to any protected service through a firewall or other access control device. Port knocking is useful for enhancing security because anyone who casually scans the target system will not be able to tell that there is any server listening on the ports protected by the knock server. Port knocking is not designed to provide bullet-proof security, and, indeed, replay attacks can easily be leveraged against a port knock server in an effort to masquerade as a legitimate client. However, there are several techniques for obfuscating port knock sequences through timing requirements, multiple protocols, passive fingerprinting of knock client operating systems, and encryption in order to make knock sequences more resistant to replay attacks. Fwknop is a complete port knocking implementation based around iptables, and supports multi-protocol knock sequences (shared or encrypted) along with passive OS fingerprints derived from p0f.

REFERENCES

1. M. Krzywinski, “Port Knocking: Network Authentication Across Closed Ports,” *SysAdmin Magazine*, vol. 12, no. 6 (June 2003).

2. J. Beale, “‘Security Through Obscurity' Ain't What They Think It Is” (2000): http://www.bastille-linux.org.

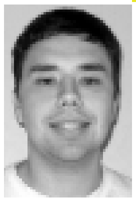
3. ICMP is implemented strictly as a network layer protocol and hence has no concept of a transport layer port number. However, the mere presence of ICMP echo requests can be made significant in terms of what a knock server expects to see, and thus adds an additional dimension to a port knock sequence.

4. See David Worth, “Cryptographic One-Time Knocking: Port Knocking Done Better”: http://www.hexi-dump.org/bytes.html.

5. The original p0f was developed by Michal Zalewski and is available for download from http://lcamtuf.coredump.cx/p0f.shtml.

STEVEN ALEXANDER

improving security with homebrew system modifications



Steven is a programmer at Merced College. He has been using FreeBSD since version 2.2.6 and still loves it.

alexander.s@mccd.edu

IN THIS ARTICLE I DISCUSS THREE modifications I've developed for FreeBSD. The first modification is a variant of the MD5-crypt mechanism, which uses an increased number of iterations in the internal loop of the crypt function. It also hashes in a constant string during the first iteration of the core loop. Increasing the number of iterations causes password hashing to require more computation time. This should not significantly impact most systems because they don't spend much of their time authenticating users. An attacker, on the other hand, wants to be able to guess millions of possible passwords per second. The attacker's efforts can be severely impacted. The constant string helps to prevent the use of standard password-cracking tools.

Another modification that I've developed is for the gcc compiler (version 2.9.5) as distributed with FreeBSD 4.8–4.10. This modification should work on other operating systems. The change I've made adds two new compile-time options to gcc. One option randomly adds up to 1 megabyte to the stack size of the function main(). The other option adds up to 16KB to the stack size of all functions. The first option is enabled by default. The second option is disabled by default and should be used very sparingly as it can have severe consequences in the way of wasted memory. Changing the stack layout of a program can defeat many buffer overflow exploits. This technique was introduced by researchers at the University of New Mexico.[1] An attacker who can tailor tools for your system will be able to defeat this defense.

More advanced randomization techniques than those used by Forrest et al. have been developed.[2] Run-time randomization of a program's memory layout is stronger but can have negative performance consequences. Load-time stack randomization is also stronger, since it is dynamic, and is currently available in RedHat Linux, OpenBSD, and PaX. I've included a patch, below, to add load-time stack randomization to FreeBSD.

For more information on buffer overflows and protection mechanisms, see the SmashGuard buffer overflow page at Purdue University.[3] More advanced memory randomization and protection measures are available with PaX and OpenBSD.[4]

The crypt modification provides hard security in that the increase in difficulty for an attacker to mount an offline password guessing attack is absolute as long as no serious cryptanalytic breakthrough is achieved against MD5-crypt. It also provides a soft measure of security through obfuscation. The modified crypt mechanism will be incompatible with other systems and standard password-cracking tools. A knowledgeable attacker can modify his or her tools to suit your system and try an offline attack, even though, because of the increased computation time, it will be less likely to succeed. Attackers who do not understand the need to modify their tools or are unable to do so will have no chance of success.

The security provided by load-time stack randomization is much more solid than that provided by compile-time randomization. In the latter case, an exploit simply needs to be tailored to a given system to work on that system. The reason it is useful is that many attackers don't have the skills or access to a particular target system needed to tailor an exploit to that system. On the other hand, load-time randomization introduces the property that many exploits will not work except by brute force, even if the attacker has access to the compiled program.

Most attackers are not expert programmers or security gurus, but tend to be kids or disgruntled employees (current or former). These attackers depend on tools that are developed by more skilled programmers. The programmers who write these tools make assumptions about the conditions of the target system. If these conditions do not hold, the tools will fail. This enables us to use both diversity and randomization for positive gain.

Randomizing Stack Sizes (Compile-Time)

In order to make gcc add a small random buffer to the top of a function's stack, I read in a 32-bit number from `arc4random()` and mask it to get the size I want. I then modify the frame offset in `init_function_start` by that many bytes and use two new compiler flags that control whether to modify `main()` and/or all other functions. The flags can be invoked using `-f[no-]randomize-stack-main` and `-f[no-]randomize-stack-all`. The latter is disabled by default. Several machines have been rebuilt with this patch in place and have been running without any problems for several months. I'm also running ProPolice/SSP on some of these machines and have not had any problems.[5] Use ProPolice (or StackGuard)—you'll be happier for it. If I've done anything taboo, I hope some of the gcc experts out there will clue me in. The gcc source lives in /usr/contrib/gcc .

function.c

In function.c, `init_function_start` requires modification. Changes are shown in bold.

```
   . . .
  void
  init_function_start (subr, filename, line)
         tree subr;
         char *filename;
         int line;
  {
  int random_dword = 0;
   . . .
  /* We haven't had a need to make a save area for ap yet.
*/
  arg_pointer_save_area = 0;
  /* No stack slots allocated yet. */
  if(flag_randomize_stack_all ||
(flag_randomize_stack_main && \
      (strcmp(current_function_name,"main")==0) ) )
  {
       random_dword = arc4random();
       if(strcmp(current_function_name,"main")==0)
            random_dword = random_dword & 0x000ffffc;
       else
            random_dword = random_dword & 0x00003ffc;
#ifdef FRAME_GROWS_DOWNWARD
       frame_offset = -random_dword;
#else
       frame_offset=random_dword;
#endif
     }
     else          /* no randomization */
     {
          frame_offset = 0;   /* Original code. If you
                  keep an extra frame_offset = 0,
                  the code won't work. */
     }
```

flags.h

The following entries must be added to the end of flags.h:

```
...
/* Nonzero means use stack randomization for main() */
extern int flag_randomize_stack_main;
/* Nonzero means use stack randomization for all functions */
extern int flag_randomize_stack_all;
```

toplev.c

These flags are then defined and set in toplev.c:

```
   . . .
  int flag_no_ident = 0;

  /* Nonzero means randomly increase the stack space used
by main by up to 1 megabyte */
  int flag_randomize_stack_main = 1;
  /* Nonzero means randomly increase the stack space used
by all functions */
  int flag_randomize_stack_all = 0;
   . . .
  {"ident", &flag_no_ident, 0,
     "Process #ident directives"}  ,
  {"randomize-stack-main", &flag_randomize_stack_main, 1,
     "Enable stack randomization for main" },
  {"randomize-stack-all", &flag_randomize_stack_all, 1,
     "Enable stack randomization for all functions" }
};
```

To rebuild gcc you should:

1. `cd /usr/src/gnu/usr.bin/cc`
2. `make obj`
3. `make depend`
4. `make all install`

Afterwards, gcc will pad the stack for main() on all newly compiled programs by default. This can be turned off by using the option -fno-randomize-stack-main. Optionally, padding can be used on all functions by specifying -frandomize-stack-all. This flag can impose a very large overhead, particularly on programs that use recursive functions. Enable it at your own risk.

The entire system can be recompiled using stack padding (for `main()` only) by:

1. `cd /usr/src`
2. `make buildworld`
3. `make buildkernel`
4. `make installkernel`
5. `reboot`
6. `make installworld`
7. `reboot`

Randomizing Stack Sizes (Load-Time)

I have also implemented load-time stack randomization by modifying /usr/src/sys/kern/kern_exec.c. The changes are minor and similar to those used in the gcc patch. In `exec_copyout_strings`, the `vectp` pointer, which becomes our stack base, is modified. This modification has been tested on FreeBSD 4.8–4.10 and 5.2.1.

This feature can be defeated by brute force or possibly in conjunction with a format string attack. In the case of brute force, the process is noisy and the attack can be stymied using techniques such as those in SegvGuard for Linux.[6] Unfortunately, such a tool is not currently available for FreeBSD. Still, load-time randomization is more difficult to defeat than compile-time randomization, as it is dynamic and an attacker must brute-force the stack addresses rather than simply analyzing the binary. Format strings can be used to defeat this technique but only under particular circumstances.

```
 . . .
#include <sys/libkern.h>
 . . .
register_t *
exec_copyout_strings(imgp)
  struct image_params *imgp;
{
 . . .
/* local variables */
 . . .
  int random_offset;
 . . .
  /*
   * The '+ 2' is for the null pointers at the end of each
of the arg and env vector sets
   */
    vectp = (char **)
      (destp - (imgp->argc + imgp->envc + 2) *
sizeof(char*));
  random_offset = arc4random();
  random_offset = random_offset & 0xffffc;
  vectp-=random_offset;
```

The kernel needs to be rebuilt to use the new changes:

1. `cd /usr/src`
2. `make buildkernel`

3. `make installkernel`
4. `reboot`

Creating a New Crypt Mechanism

A few months ago, I was re-reading parts of *Practical UNIX and Internet Security*[7] and noted the authors' suggestion that system administrators modify the crypt routine on their UNIX systems to loop more than the standard 25 times, in order to prevent attackers from using a standard password cracker. I decided to implement this on some of my systems. This modification works on FreeBSD 4.8–4.10 and 5.2.1. The existing source code for 5.2.1 looks slightly different, but the changes are the same.

If I were simply to change the existing source code and recompile, all the accounts on my systems would stop working. That is why FreeBSD allows new crypt mechanisms to be added without replacing the original mechanisms. All new passwords are hashed using the mechanism that is configured in login.conf.

Rather than modify the old DES-based crypt mechanism, I modified a copy of the MD5-based crypt mechanism, which is much stronger. MD5-crypt was designed by Poul-Henning Kamp and uses Ron Rivest's MD5 hash algorithm.[8] I do not suggest arbitrarily modifying the crypt mechanisms unless you have real cryptographic expertise, as you may inadvertently weaken the algorithm. I have only increased the number of iterations of the algorithm and hashed in a constant value; everything else is intact.

Passwords that are hashed on your system using the new crypt mechanism will not be breakable with an unmodified password cracker. To make the job of an attacker more difficult, back up and remove the libcrypt source code from your servers after the new libcrypt has been installed. An attacker can still analyze the binary code to find out how it was modified, but many attackers do not have these skills. If an attacker is unable, or unknowingly neglects, to do this, his or her offline attack will not succeed.

On FreeBSD and many other systems, non-root users are not able to see the stored password hashes. These hashes are still valuable to an attacker. Many attackers copy the password file after a break-in so that the passwords can be tried on related systems to which the attacker does not have access or reused on the same system if the hole the attacker used to break in is patched.

Adding a new crypt mechanism to FreeBSD turns out to be pretty easy. The source for libcrypt is located at /usr/src/lib/libcrypt. Three files need to be changed to create a new mechanism: crypt.c, crypt.h, and Makefile. A file must also be created that contains your new mechanism.

crypt.h

The header file crypt.h contains the prototypes for the different crypt mechanisms. You can name your new mechanism whatever you like; mine looks like this:

```
 . . .
char *crypt_md5_local(const char *pw, const char *salt);
 . . .
```

crypt.c

crypt.c contains a data structure named `crypt_types` that contains the name of the mechanism, the function to call to use the mechanism, and the magic value that is prepended to passwords that use this mechanism. My entry to the list looks like:

```
 . . .
```

```
  {
    “md5local”,
    crypt_md5_local,
    “$4$”
  },
  {
    NULL,
    NULL
  }
 . . .
```

crypt-md5-local.c

In order to create a new crypt mechanism, I copied the file crypt-md5.c to crypt-md5-local.c. You must modify the new file slightly. Rename the function `crypt-md5` to `crypt-md5-local`. Near the beginning of the function, the magic value should be changed from `$1$` to `$4$`. The magic string `$3$` is not used in FreeBSD 4.x but is used in FreeBSD 5.x for the NT hash algorithm. Your changes should look like this:

```
 . . .
char *
crypt_md5_local(pw, salt)
  const char *pw;
  const char *salt;
{
  static char*magic = “$4$”;   /*
                    * This string is magic for
                    * this algorithm. Having
                    * it this way, we can get
                    * better later on.
                    */
 . . .
```

Further down in the code, a `for` loop iterates 1000 times to form the core of the MD5-crypt mechanism. You can replace the value 1000 with anything you like. Increasing the number significantly will make password cracking very difficult; however, increasing it too much could slow the system unnecessarily. If, for some reason, you need password database information to be interoperable on multiple systems, each system will need to use the same value in its modified crypt mechanism. Here, I have changed the number to `8000`. I also have hashed in the constant string `mercedcollege`; this prevents an attacker from performing an offline password cracking attack without modifying his or her tools to suit. Modify this string to something of your own choosing.

```
 . . .
  for(i=0;i<8000;i++) {
    MD5Init(&ctx1);
  if(i==0)
    MD5Update(&ctx1,”mercedcollege”,strlen(“mercedcol-
lege”));
    if(i & 1)
      MD5Update(&ctx1,pw,strlen(pw));
 . . .
```

Makefile

The name of the file that contains your new crypt mechanism must be included in Makefile:

```
  SRCS=    crypt.c crypt-md5.c crypt-md5-local.c md5c.c
misc.c
```

/etc/login.conf

After these changes are made be sure to `make` and `make install`. /etc/login.conf can be modified to use this new method as the default. The entry should look like:

```
  :passwd_format=md5local:\
```

Afterwards, you must run `cap_mkdb /etc/login.conf`.

To install the new libcrypt:

1. `cd /usr/src/lib/libcrypt`
2. `make`
3. `make install`
4. `reboot`

REFERENCES

1. S. Forrest, A. Somayaji, and D. Ackley, "Building Diverse Computer Systems," *Proceedings of the 6th Workshop on Hot Topics in Operating Systems* (1996): http://www.cs.unm.edu/~immsec/publications/hotos-97.pdf.

2. Monica Chew and Dawn Song, "Mitigating Buffer Overflows by Operating System Randomization," Tech Report CMU-CS-02-197 (December 2002); PaX, http://pax.grsecurity.net/.

3. See https://engineering.purdue.edu/ ResearchGroups/SmashGuard .

4. See http://pax.grsecurity.net/, http://www.openbsd.org.

5. Hiroaki Etoh, "ProPolice: GCC Extension for Protecting Applications from Stack-Smashing Attacks," IBM (April 2003): http://www.trl.ibm.com/projects/security/ssp/.

6. Nergal, "The Advanced return-into-lib(c) Exploits: PaX Case Study": http://www.phrack.org/phrack/58/p58-0x04.

7. Simson Garfinkel and Gene Spafford, *Practical UNIX and Internet Security* (Sebastopol, CA: O'Reilly, 1996).

8. Ron Rivest, "The MD5 Message-Digest Algorithm," RFC 1321 (April 1992).

DARIO FORTE, DAVIDE BERTOLETTI, CRISTIANO MARUTI, AND MICHELE ZAMBELLI

Sebek Web "ITA" interface

AN ALTERNATIVE APPROACH

Dario Forte, CISM, CFE, is adjunct professor of incident response and forensics at the University of Milano, Crema Campus. He is also the founder of the Incident Response Italy Project and the coordinator of the Italian Honeynet Project and is the president of the European Chapter of the HTCIA.

dario.forte@acm.org

Davide Bertoletti, Cristiano Maruti, and Michele Zambelli are graduate students at the University of Milano at Crema and are part of the Incident Response Italy team and the Italian Honeynet Project. Their research interests focus on security and incident response.

IN 2003, A GROUP OF TEACHERS AND undergraduates at the University of Milano at Crema started a project called Incident Response Italy (IRItaly), whose aim was to provide guidelines for incident response and forensics. After five months, the same group founded the Italian Honeynet Project (IHP), which became part of the Honeynet Research Alliance. Since then the IHP has worked on many tasks, like the beta and national deployment of the Honeynet Security Console (from Jeff Dell). The IHP is now working on the development side as well, by contributing to the Sebek Project.

Sebek, a tool created by Ed Balas of Indiana University, is basically a piece of code that lives entirely in kernel space and records either some or all of the data accessed by users on the system. It has the ability to record keystrokes from a session that is using encryption, recover files copied with SCP, capture passwords used to log in to a remote system, recover passwords used to enable Burneye-protected binaries, and accomplish many other forensics-related tasks. You will find more information on the tool in the papers mentioned in the bibliography. In this article we'll discuss our approach to the Web interface.

Our Work

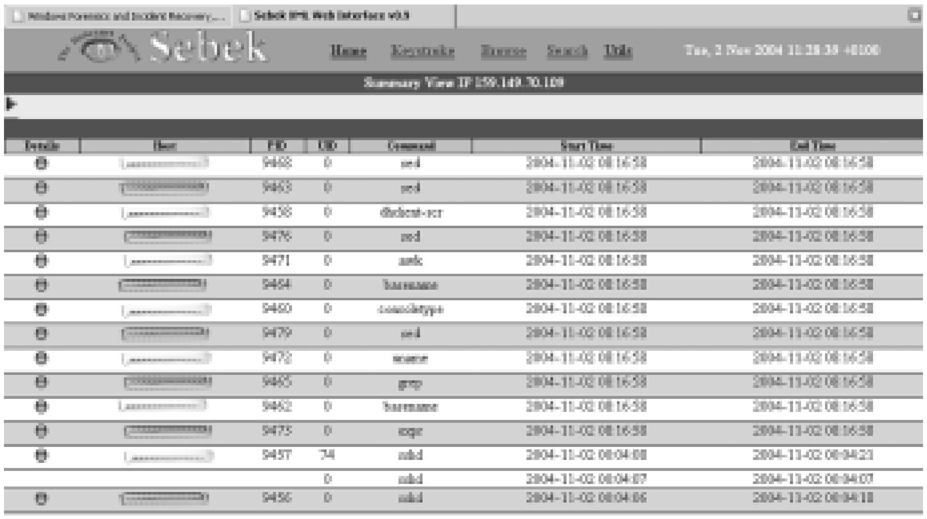
Although the current version (0.8) of the Sebek Web Interface is stable and complete, we have added some additional features that make it even more useful and applicable in a broader variety of situations. The new features are the following:

- XML output
- WAP interface
- Dump database
- Paging

XML OUTPUT

We have written a Web application that displays information collected by Sebek. The application is written entirely in PHP and returns the information as an XML document. We rewrote the tool to output data in XML because it is more flexible than the HTML format previously used. We have also added capabilities to view the status of monitored hosts via cellular phone or WAP-enabled device. We have chosen an XML, Web-based application for a number of reasons:

- XML is becoming the database language for the Web.
- XML is the interchange mechanism between applications.
- XML cleanly separates presentation layers from data layers.



Summary View IP 159.149.70.109

Details	Host	PID	UID	Command	Start Time	End Time
●	[illegible]	9468	0	sed	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9463	0	sed	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9458	0	dhclient-scr	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9476	0	sed	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9471	0	awk	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9464	0	basename	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9460	0	consoletype	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9479	0	sed	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9472	0	uname	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9465	0	grep	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9462	0	basename	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9475	0	expr	2004-11-02 08:16:58	2004-11-02 08:16:58
●	[illegible]	9457	74	sshd	2004-11-02 08:04:08	2004-11-02 08:04:23
			0	sshd	2004-11-02 08:04:07	2004-11-02 08:04:07
●	[illegible]	9456	0	sshd	2004-11-02 08:04:06	2004-11-02 08:04:18

Figure 1. Sebek in Browse Mode

Basically, the architecture works as follows:

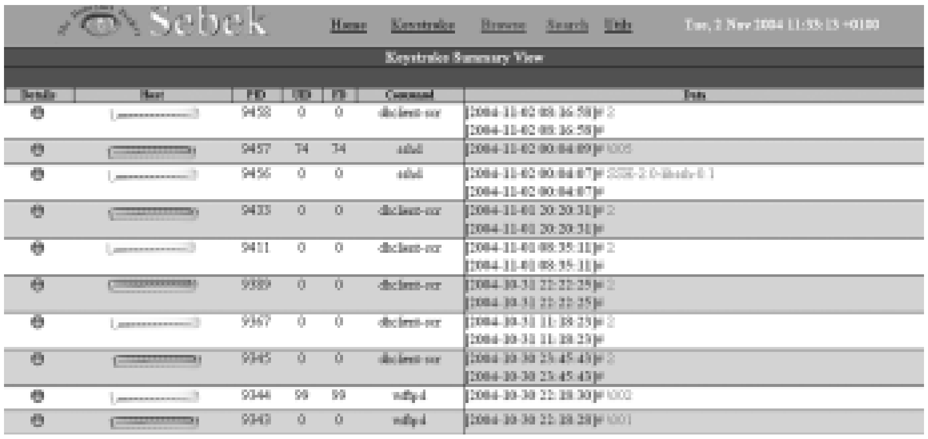
The application creates a connection with the remote database, gets and modifies the data, and sends the data back as an XML file. Data is displayed as HTML pages using an XSL stylesheet. The application is also usable in a wireless environment via WAP technology.

The XML file has a two-part structure. The first is located in the header tag and contains information concerning the header of the HTML page; the second, placed between Sebek tags, contains data obtained from the database. Both of these parts are placed between the root tags.

```
<?xml version=”1.0” ?>
<root>
<header>
.
.
.
.
</header>
<sebek>
.
.
.
.
</sebek>
</root>
```

Let’s have a look at the most common tags in the document:

- `root`: the main tag, within which the other components of the document are placed
- `header`: contains information about the page header
- `sebek`: contains the data created by a database query
- `read data`: contains some information about the database fields



Details	Host	PID	UID	FD	Command	Data
●	[illegible]	9458	0	0	dhclient-scr	[2004-11-02 08:16:58]# 2 [2004-11-02 08:16:58]#
●	[illegible]	9457	74	74	sshd	[2004-11-02 00:04:09]# 1005
●	[illegible]	9456	0	0	sshd	[2004-11-02 00:04:07]# SSH-2.0-libssh-0.1 [2004-11-02 00:04:07]#
●	[illegible]	9433	0	0	dhclient-scr	[2004-11-01 20:20:31]# 2 [2004-11-01 20:20:31]#
●	[illegible]	9411	0	0	dhclient-scr	[2004-11-01 08:35:11]# 2 [2004-11-01 08:35:11]#
●	[illegible]	9389	0	0	dhclient-scr	[2004-10-31 22:22:25]# 2 [2004-10-31 22:22:25]#
●	[illegible]	9367	0	0	dhclient-scr	[2004-10-31 11:18:23]# 2 [2004-10-31 11:18:23]#
●	[illegible]	9345	0	0	dhclient-scr	[2004-10-30 23:45:43]# 2 [2004-10-30 23:45:43]#
●	[illegible]	9344	99	99	wftpd	[2004-10-30 22:18:30]# 1002
●	[illegible]	9343	0	0	wftpd	[2004-10-30 22:18:28]# 1001

Figure 2. Sebek in Keystroke Mode

There is a relationship between data returned by the Web interface and the data stored in the database. Other data are obtained via aggregation functions or similar operations:

- `ip addr:` IP address of the port where the commands were issued
- `insert time`: insert time of the information in the database
- `command`: command executed by the intruder on the compromised host
- `time`: current host time
- `fd`: file descriptor
- `pid`: process ID
- `uid`: user ID
- `length`: byte length of the recorded activity
- `data list`: a summary list of the keystrokes executed by the intruder
- `rec num`: total number of inserted records
- `start time`: start command time
- `end time`: end command time

In addition to the tags we have explained above, there is another type of tag with a different kind of function:

- `link`: creates a link to a different page. The link can be text type or image type. In addition to the type of link, defined by the `aspect` attribute, there is an additional attribute. The `page` attribute is used to define the link, the frame in which to open the new page, and possibly other parameters.
- `now`: contains information about local server date and time.
- `space`: defines an empty area.

WAP INTERFACE

It is possible to get information about our honeynet via a WAP device, without using a networked PC. By means of a device which supports WAP technology, we are able to verify at any time and from any location whether the network is under attack.

The most useful part of the WAP interface is that it provides a summary page containing the monitored hosts, the number of records for each host, and the latest update. Only these few items were chosen for display because of the limited computational and graphics capabilities of WAP devices. The details of teh situation are reported through the HTML application. In any case, the summary page provides a useful snapshot of the monitored hosts.

It is also possible to create a dump of the database from the WAP device, which may be handy if we are not connected through a "normal" network device.

DUMP DATABASE

After some days of network activity, we observed an overload of the Sebek MySQL database. We therefore created the Utilities section of the interface to allow a dump of the current database. The operation is catalogued with dump date and time information, and an empty database is created. This brilliant solution allows for faster browsing of the collected data entries and a reduction in transmission delay. It is also possible to activate a database dump via a WAP interface.

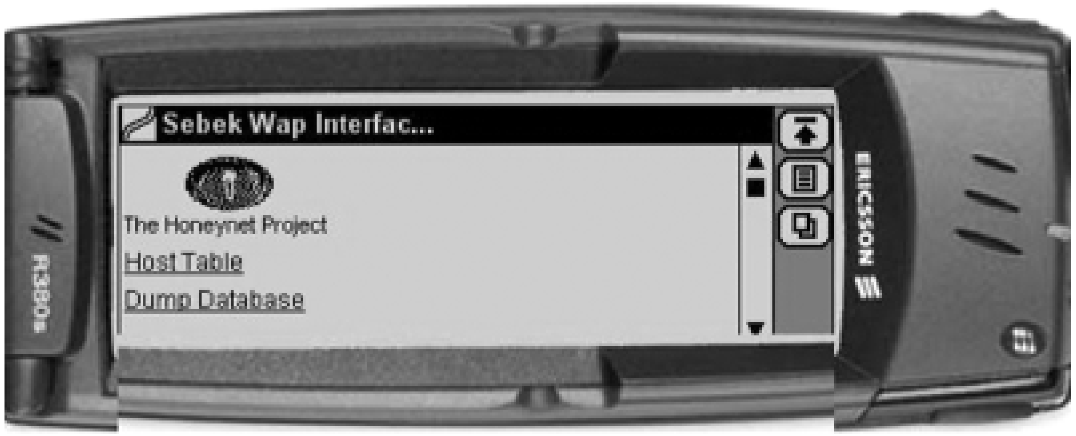


Figure 3. WAP Interface to Sebek

Further Developments

The Sebek Web ITA Interface can be downloaded from http://www.honeynet.it. Comments and feedback are welcome. Of course, it is just an experiment, but we are pretty confident that this tool can be a valid alternative to the current version of the interface. Meanwhile, Sebek is going to undergo some major changes. Ed Balas has presented the next version of Sebek, which will include a new interface, and the Italian Honeynet Project group was added to the developer team. Sebek and its implementations are proof that the Honeynet Project is maturing rapidly and effectively.

Acknowledgments

The tool was developed by the Italian Honeynet project, a group of people working with the University of Milano at Crema. Special thanks to Michele Zambelli, Cristiano Maruti, and Davide Bertoletti, who worked on the source code and the architecture.

REFERENCES

Ed Balas, “Honeynet Data Analysis: A Technique for Correlating Sebek and Network Data,” *Proceedings of the Digital Forensic Research Workshop*: http://www.dfrws.org.

Ed Balas, “Know Your Enemy: Sebek, a Kernel-Based Data Capture Tool”: http://www.honeynet.org/papers/sebek.pdf.

Dario Forte, “The Art of Log Correlation: Tools and Techniques for Digital Investigations,” *Proceedings of the Information Security South Africa 2004 Conference*: http://www.dflabs.com/images/Art_of_correlation_Dario_Forte.pdf.

The Italian Honeynet Project: http://www.honeynet.it.

JENNIFER S. GRANICK

strike back

Jennifer Stisa Granick joined Stanford Law School in January 2001 as a lecturer and is Executive Director of the Center for Internet and Society (CIS). She teaches, speaks, and writes on the full spectrum of Internet law issues, including computer crime and security, national security, constitutional rights, and electronic surveillance, areas in which her expertise is recognized nationally.

jennifer@granick.com

SEVERAL COMPANIES ARE DEVELOPING interesting new programs system administrators can use to disable remote machines that are sending damaging packets to their computer systems. These technologies—often called "strike back" or "active defense"—foment a lot of interest among those of us fed up with an avalanche of unending worms and viruses.

But are strike-back technologies legal? The law simply hasn't developed to the point where there's a clear answer, but sysadmins resorting to strike-back are playing with legal fire.

strike-back technologies are designed to locate the source of unwanted or harmful Internet connections and shut those machines down. For example, one such program responds to the Code Red worm by identifying the machine sending the worm, and using a back door left by the worm itself to install and execute code that stops the attacking machine from transmitting the worm.

Many critiques of strike-back focus on the risk of retaliating against the wrong machines. IP spoofing can mask the true origin of unwanted packets. Also, an attack may come from an unwary third party's machine that is itself the victim of an attack. Disabling that machine may cause the innocent owner serious problems. At the very least, the owner will realize that the strike-back program has altered his or her system and will need to investigate to determine exactly what happened.

Even if a strike-back program accurately targets the source of the attack, state and federal laws prohibit unauthorized access to and modification of networked computers. These laws not only outlaw the transmission of worms and viruses but also prohibit victims of attacks from themselves intruding on their attacker's systems, regardless of motive. Any unauthorized access to a networked computer that causes damage of $5000 or more (which includes the costs of investigating the access) violates federal law. "Unauthorized access" currently means connecting to the computer without the permission of the system owner. Most state laws prohibit unauthorized access whether or not it causes damage. Users of strike-back technology may be buying themselves a civil suit or, worse, criminal prosecution.

In time, legal rules may embrace strike-back. Congress could decide to give system owners the right to disable attacking machines, as it recently proposed doing for intellectual property owners who discover their copyrighted information on peer-to-peer networks. Or judges confronting strike-back cases may decide to extend traditional legal excuses such as self-defense or defense of others to this new situation.

The doctrine of self-defense or defense of others permits the use of otherwise illegal force to prevent harm to oneself or to others under certain circumstances. The precise definition of self-defense differs from state to state, but as a general rule, self-defense applies only if the force used to repel the harm is necessary, reasonable, and proportional. As applied to strike-back, a judge might think disabling a system from sending Code Red packets is self-defense, but completely paralyzing the system or reformatting the hard drive is not.

The excuse of self-defense usually applies only if you have no other means of protecting yourself. Some states even require you to retreat if possible, to leave the scene of the problem, before resorting to self-defense measures. There are usually alternatives to strike-back, whether it's taking your system offline (a digital form of retreat, perhaps), firewalls, or comprehensive patching. Perhaps a court will find that self-defense is never a valid excuse, because the first line of defense is to secure the system properly, not to strike back against attackers.

It's folly to ask judges or juries to calculate whether a digital retaliation is necessary, reasonable, and proportional when the security community itself doesn't yet agree on best practices. But in light of the interest in strike-back technology and the eagerness of sysadmins to deploy it, it won't be long before judges have to decide whether strike-back is self-help or vigilantism.

MICHAEL B. SCHER

SSO/RSO/USO/ oh no?

Mike Scher makes his way in life as a security policy and architecture consultant in Chicago. An attorney, anthropologist, and security technologist, he's been working where the policy tires meet the implementation pavement since 1993.

mscher@cultural.com

THE RECURRENT CORPORATE DRIVE TO create[1] a reduced, single, unified, or otherwise simpler sign-on process is experiencing a new growth phase, based on password authentication. The rush to use passwords, indeed to use one password for everything, should give IT security personnel pause. The move to use one password for everything strikes me as significant backsliding. It may be that I differ from many of my contemporaries in that I think even "good" passwords are not always a good idea.

The only thing that's wrong with password authentication is that it uses passwords.—Me

It's one thing to use passwords and a unified login portal when you have 1.5 million customers using your Web-based services, but it's quite another to clump an employee's corporate, financial, business, and administrative access behind a single password. That single password becomes everything, a valuable steppingstone taking the single-sign-on system into the brave new world of "single break-in."

When passwords are strong, they can become difficult for users to manage. Forcing users to maintain six or more passwords with variant rules, change periods, and so forth won't increase real-world security. When we do find passwords manageable, it's often because we have created weak, predictable, guessable, crackable, or easily stolen passwords. When we succeed in making strong passwords manageable, we have usually done one of three things: tucked them all behind another password (e.g., cryptographic password safes, certificate stores), placed them all someplace risky (e.g., wallet, keyboard tray, text file, Word doc), or made a central authentication system of some sort that lets users just use one or a couple of passwords for everything.

Let's try to define some terms:

- *Unified sign-on:* Passwords are synchronized or authentication is centralized. Think RADIUS servers with passwords, standard Microsoft domain or Active Directory authentication, Sun's NIS/NIS+, central strong-authentication servers, or the many password-synchronizing programs that don't require a central authentication server (but which do require a system to coordinate your global password space).
- *Single sign-on:* You log in to one application or host, and other items know you're logged in, because you send some special blob of data to them that proves it. Each device validates the data you send, either by passing it to a central authentication system or by

cryptographically validating it. SSO implies a flat or tiered trust model, which must be well documented for it to be effective. Full SSO is often deployed with Web-based applications, using browser cookies or complex URLs to "pass the hash," by way of products like RSA Cleartrust and Entrust, and with many Kerberos implementations, Microsoft's included.[2]

- *Reduced sign-on:* What most companies get when attempting to implement single sign-on. At best, systems that share common authentication are grouped by security level and purpose, so the user has a handful of authentication items that "do it all" ranked by security, each group its own trust realm. At worst, there is no security coordination; all systems that could be put on SSO are on one SSO or RSO system, the rest aren't. Then, for example, significant blurring of security realms may occur: Internet-facing, cleartext Web interfaces may take the same password as the corporate benefits and health insurance service system's SSL interface, two IP addresses over.

TYPES OF REDUCED- OR SINGLE-SIGN-ON SYSTEMS

- Password synchronization/coordination systems
- Authentication store consolidation (AD, LDAP, NIS/NIS+)
- Portal-based SSO and authentication gateways (including transparent man-in-the-middle authenticators)
- Token/cookie-enabled "start pages" (sometimes portals)
- Token/cookie distribution centers (Cleartrust/Kerberos, etc.)

Risks

The risks presented by SSO systems stem from two significant audit and coordination issues: authentication credentials and user IDs.

AUTHENTICATION CREDENTIALS ARE LIKE EVIDENCE

I tend to think of authentication as an evidence problem: "Does what the user is providing meet the level of evidence that we require to prove they are who they say they are?" Passwords, for example, are weak as evidence: They can be stolen without the user's knowledge, copied, intercepted, guessed. The stronger the passwords are and the better the mechanisms over which they travel and their compliance with good password policy, the better they are as evidence. Good password policy, auditing, and a well-designed sign-on architecture can significantly boost the efficacy of password-based security.

Nevertheless, on their own, passwords don't really rise above what the legal world might term a "preponderance of the evidence" (essentially, "more likely than not") in terms of evidentiary value. In the effort to make them stronger, we combine them with other bits of evidence: physical access, IP addresses (e.g., the finance service center segment), digital certificates (which are used for most authentication purposes as large passwords protected by small passwords), soft tokens, hardware tokens, time of day, and so on. Just as with evidence in a trial, the more evidence that points to the same conclusion, the more likely we are to believe the conclusion likely is true. Indeed, "more likely than not" on its own is not so bad. It's the standard of proof required in many kinds of civil suit, and it's the standard that (everything taken in the best light for the plaintiff or prosecutor) must be met before a case may go to trial.

RATE YOUR SYSTEMS, ORGANIZE YOUR USERS

For some purposes passwords alone may be "good enough," but remember that we are talking about a shared authentication framework: The selected authentication mechanism is going to be used for many systems. The level of protection required may vary from one system to another.

Thus, before a company implements a new password-based sign-on regime, it should consider a company-wide effort to rate and rank hosts and applications by business-criticality, regulatory requirements, and susceptibility to abuse via stolen authentication credentials. Next, the company should try to organize users' network, host, and application login names into a unified or coordinated scheme. For real SSO, a company-wide identity management scheme helps administrators select the systems to which a given user should have access and the rights of each class of user, and allows ready changes in user authentication rights as responsibilities change. To ensure proper scope, the company must not only define what systems are critical, but also the kinds of system that are to be "in scope" for centralized authentication. At many organizations, Sarbanes-Oxley compliance initiatives are already producing a set of critical systems and authorized users, a first step on the path to real identity management.

The risks of not ranking systems and applications are significant. A system that ought to use a higher-strength credential could be tied into a basic password-based sign-on regime. A server that authenticates "in the clear" (internally or even over the Internet) could be tied in, exposing passwords used for more sensitive applications. Policies that are well defined, user education, and a good audit process are essential to strengthen a password-based sign-on regime.

PASSWORDS AND UNIFIED-SIGN-ON SYSTEMS: LIKE KETCHUP AND CHOCOLATE CAKE?

Depending on the company, information that is sensitive, business-critical, trade-secret, or regulated could require stronger credentials than passwords alone. Some authentication requirements (authorizing transactions or launching events that will have major ramifications) may require standards of proof that come close to "beyond a reasonable doubt." We can achieve those higher levels by using multiple factors that together rise to the level we need. Thus if we can ensure that a password has sufficient strength, is regularly audited, and is never used outside the corporate network, even for remote access, we can have a higher level of trust in it. We could perhaps trust such a password for use on more sensitive systems. Again, policy, education, and audit are key to using passwords across the enterprise, with or without a fancy sign-on system.[3]

RISKS FROM REDUCING/COORDINATING SIGN-ON

- Payoff from password-guessing attacks (dictionary and brute force) is increased. Think single break-in rather than single sign-on.
- Password theft impact increases.
- Coordinated DoS from bulk lockouts becomes a risk.
- SSO may be provided to system-level accounts or to user IDs that have varied access levels across the enterprise, placing the wrong level of safeguard on critical accounts/systems.
- Global authentication credentials may be exposed through integration with external, non-SSL Web systems or through use on third-party, insecure hosts (kiosks and the like); users may also use the same password on third-party systems, from banking to blog sites and beyond.
- Remote authentication gateways and VPNs may use the same, easily stolen authentication credential, making single break-in a real possibility.

Benefits

Don't get me wrong: A well-implemented SSO system can actually increase security, but it's going to take some work to get there. The mechanism behind single sign-on that lets the server know the user has already logged in can be more secure than the basic username/password mechanism an integrated application used to use, as with using Kerberos tickets instead of user/pass over unencrypted channels. Policy at the company may already be ill-enforced—users regularly selecting weak passwords, leaving them in text files on their laptops, etc.—such that consolidation to just a few passwords may allow better policy enforcement along with increased user convenience.

When combined with a well-executed identity management program, the SSO system may well provide some cost savings. Old accounts on per-seat systems can be removed in a timely way; help-desk costs for password management may go down; new employees may find all their accounts up and working right away; new services can be rolled out on new systems without having users go through the "new password" process. One must take care, however, to rate these savings in a conservative way. Costs will not entirely disappear, and other issues will rise to take their place.

BENEFITS

- Convenience to both end users and developers
- Account provisioning and removal efficiency, and cost savings (including better oversight of per-seat license costs)
- Centrally enforced, consistent authentication policies
- Easier implementation of identity management systems, if the company is trying to get a grip on distributed systems
- Auditing and anomaly-based alerting for application and host login activity

Real-World Costs

Most of the costs are predictable, up-front costs to make the new authentication regime a reality. Do not, however, forget that there will be ongoing costs in addition to the systems themselves. Legacy systems in particular may house critical applications that will require careful, custom coding to enable integration without simultaneously creating security exposures. Depending on the sign-on system implemented, the company may need to step up its audit activities, policy enforcement, and change-control procedures. Lost or forgotten authentication credentials mean the user can do no work until the authentication is replaced or a temporary authentication credential is assigned (one-use password, for example). The increased need to assign temporary credentials so that users can get back up and working leads to a need for superior controls against social engineering. The SSO core systems become critical to the workday, so there need to be redundant systems in resilient (and perhaps geographically diverse), physically secure environments with fault-tolerant networking to the rest of the company.

COSTS

- Fault-tolerant systems in fault-tolerant environments
- Recurring software licenses (and per-user licenses)
- Software integration costs (internal staff and consultants)
- Any hardware/license costs for strong authentication devices
- Heightened audit and staff education costs in some environments

Real-World Quagmires

I've been unfortunate enough to witness several failed unified- or single-sign-on rollouts from the perspective of an external security consultant. The process at the large organizations involved inevitably proceeded in the following manner:

1. Users complain about having a ton of passwords with conflicting creation and change policies (some of which are misguided in terms of the real risk profile).
2. Security admins crack down on users reusing passwords or using poorly constructed passwords, making the issue both an embarrassment and even more unmanageable.
3. Users complain even more; help-desk calls go up as users forget new passwords more frequently than before.
4. Management does a study and finds that over 50% of help desk calls are about password-related or account-creation issues; management may already be looking at or paying for password reset management software.
5. Management calls for SSO, arguing that it's an efficiency and cost issue.
6. IT security calls for tokens and a central authentication system.
7. A vendor-affiliated (or vendor-owned) consulting firm does a study showing how much more token-based SSO/RSO will cost than plain-password USO/SSO because of legacy system issues and per-seat costs for tokens.
8. The consulting firm makes arguments about how much more secure passwords can be with central password policy enforcement, without regard to the costs of ensuring policy compliance.
9. Someone says, "Isn't this just going back to Sun's YP/NIS?" but is quickly hauled out of the building.
10. Millions are slated for analyst input, consultants, redundant hardware, software, licenses, and implementation.
11. By the time the SSO system is up in prototype, management finds that critical legacy systems can't be integrated with it except at very high cost.
12. Other authentication systems start coming in (again). If first time here, go back to step 1 and start over. Otherwise, continue . . .
13. Eventually, wiser organizations base all new authentication on one easy-to-use token authentication system (plain RSA key fobs or SafeWord Silver tokens, for example), while integrating Web-based applications behind one or more single-sign-on portal systems.

Conclusion

Reducing the number of times and ways a user needs to sign on to applications and systems is a worthy goal. Password policies don't scale well when users are faced with six, seven, or more sets of divergent policy implementations (even where policy is consistent). However, one must take care not to assume that integrated sign-on regimes are the only way to straighten out the authentication mess. Further, one must take great care to manage expectations, risks, costs, and timelines from the outset. Set fully articulated objectives based on realistic, cost-justified goals, and it can work.

Finally, base whatever you do on public standards, either open or easily extensible ones. Adhering to standards ensures that future development and new applications cost less to integrate, and standards will meet with readier recognition from business partners, auditors, and new systems personnel. There are several well-fleshed-out standards for portal-based and central authentication integration. For example, for portal-based and Web-integrated SSO, the Liberty Alliance standard[4] (employing SAML),[5] Kerberos, and MS Kerberos[6] approaches are solid, vetted, and well documented. For centralized authentication, there are, of course, RADIUS, LDAP (and S/LDAP), MS AD, and many others.[7]

RECOMMENDATIONS

- Combine any sign-on project with an enterprise-wide identity management project. Benefit: Sarbanes-Oxley audits may be taking you down this path already. Take advantage of the work now, while it's current.

- Combine any sign-on system with strong authentication products, such as hardware tokens on an application-class or enterprise basis, to ensure that critical credentials are less prone to theft, guessing, or other compromise.
- Carefully spell out the real goals of the project. Ensure enterprise-wide buy-in.
- Ensure that most users will experience a noticeable reduction in sign-on events. The best deployments of strong authentication have user demand escalate dramatically, even with no SSO benefit, when users find that they no longer need to deal with a dozen passwords. With an SSO result, the user buy-in can thus be even greater.
- Ensure that the project has realistic cost estimates and measurable goals. Expect integration difficulties and special development and consulting costs. Target specific problems that have readily assessed costs to the enterprise today, so that the predicted cost savings can easily be substantiated.
- Base the final technical architecture on public standards that provide integration examples, reference code, and excellent documentation.

NOTES

1. Or is it re-create? See Andrew Findlay's piece on the corporate world's quest to regain the mainframe login milieu, "Regaining Single Sign-On," at http://www.brunel.ac.uk/depts/cc/papers/regaining-sso.html.

2. The Open Group has an introduction to single-sign-on concepts and the trust relationships they entail at http://www.opengroup.org/security/sso/sso_intro.htm.

3. See G. Ellison, J. Hodges, and S. Landau, "Risks Presented by Single Sign-On Architectures," at http://research.sun.com/liberty/RPSSOA/, and Avi Rubin's risk analysis of Microsoft's Passport SSO protocol at http://avirubin.com/passport.html.

4. See Project Liberty's home pages at http://www.projectliberty.org/; for a good overview of its current market position, go to http://www.eweek.com/article2/0,1759,1619564,00.asp.

5. See http://java.sun.com/features/2002/05/single-signon.html for a practical description of SAML in the SSO realm.

6. See http://www.microsoft.com/technet/security/topics/identity/idmanage/P3Intran_1.mspx.

7. Diana Kelley and Ian Poynter gave an excellent overview of single-sign-on issues at Black Hat USA 2002. Their presentation is available at http://www.blackhat.com/presentations/bh-usa-02/bh-us-02-poynter-sso.ppt, with audio at rtsp://media-1.datamerica.com/blackhat/bh-usa-02/audio/2002_Black_Hat_Vegas_V06-Diana_Kelly_and_Ian_Poynter-Single_Sign_On_101-audio.rm.

ELIOT LEAR

being awash in keys

A member of the USENIX community since 1988, Eliot Lear works for Cisco Systems, Inc., and has been their self-proclaimed corporate irritant since 1998. He's been a consumer a bit longer.

lear@cisco.com

TODAY'S CONSUMER MUST KEEP TRACK of many passwords in order to make use of online banking, commerce, and government services. Attempts to consolidate authentication methods into a single sign-on service have thus far failed. Worse, with viruses, spam, and phishing, more sophisticated authentication is demanded. That sophistication today has brought confusion to the consumer. What stops the consumer from having a single secure password? This article considers several areas of improvement the industry should consider, and we'll put forth an example of a single smartcard that could potentially provide unified authentication.

A (Very) Brief History of Success

The enterprise has largely been successful at consolidating identity. This evolution began with common host access through mechanisms such as Kerberos.[1] Common network access was then made possible through protocols and mechanisms such as TACACS and RADIUS.[2] Application integration is now following as directory services evolve. All of this is great for the enterprise, because with a single administrative function tied to a registrar or human resources function, it is possible to enable or disable a user, change user rights, and retrieve a log of user activity. This success has been made possible by a vested interest in consolidating administrative overhead costs. Unfortunately, not only are most of the mechanisms developed for the enterprise inappropriate for consumer authentication, but those products actually contribute to the individual's inconvenience. What does the consumer need? What is necessary for the e-commerce vendor?

The Phishing Example

Today most US banks use simple username/password security with one-way SSL authentication and encryption.[3] Put another way, the bank server authenticates itself to the user's browser, and in return the user sends a username and password through a form to authenticate himself or herself to the server. From a protocol standpoint, this method provides mutual authentication. It breaks down, however, at the user interface level. Published reports of US bank losses range from $500 million to $1.2 billion last year, while British banks lost over £1 billion.

In order to stem the losses, some banks have attempted to up the ante by requiring sophisticated challenge-response systems. Even these systems, however, are not impervious to attack. Consider the case in which the masquerader receives from a potential victim a username necessary for generating a challenge, sends the username to the real bank, parses the challenge, and then sends it back to the victim. In other words, these miscreants are able to effect a classical man-in-the-middle attack, all because the user didn't bother to click the little lock icon in his browser to verify that the certificate was correct. Worse, most people don't know what the correct certificate would look like.

This same bank might want to raise the stakes further by attempting to make the challenge readable only to humans (and perhaps just barely at that). Unfortunately, the same technology used by spammers and phishers is also used by legitimate screen readers for those who are visually impaired.

One approach to solve this problem is to make use of public/private key pairs in the smartcards. There are many methods to choose from, but the key point is finding where the line of trust begins and ends. SANS infection statistics claim that the average survival time of an unpatched Windows PC is 19 minutes.[4] Clearly the PC is not to be trusted. This means that the point of trust must be the smartcard itself, and that it must maintain an opaque channel through the PC to the authentication system on the other side.

Multiply the single bank login by investment houses, health insurance, travel sites, bookstores, telephone companies, not to mention an employer, so that in order to retain convenient access one needs a whole bag of hardware and a whole slew of login names. Instead, wouldn't it be nice if the user could make use of one or more identities to authenticate against any particular service?

Privacy concerns also abound. Any solution in this space will have to come to grips with the idea that vendors may wish to sell information about consumer identities. While some jurisdictions, such as the European Union, provide strong controls for consumers, others, such as the US, do not. Such correlation of information is difficult to prevent, in part thanks to HTTP and cookies. While it is unlikely that any solution in this space could help consumer privacy, one wishes not to add to the problem.

"Many Have Tried"

There have been several attempts at providing individuals with unique identities that could be used for commerce. The reasons they have failed are complex, including implementation limitations, trust of the provider, vendor costs, and competitive concerns. As we've seen, a software implementation leaves a tremendous amount to be desired. Any solution in this problem space must be widely accepted by both consumers and vendors, thus requiring that the needs of both be met. Inasmuch as credit card companies indemnify consumers from identity theft, they too have skin in this game.

There exist a number of standards in this space already. A plethora of ITU (International Telecommunication Union) standards define the interface between smartcards, computers, and identities. SASL, SSL, and TLS provide a means of transporting authentication over the network, and Mozilla provides a way to use hardware tokens. However, none of these standards has established a sufficiently trustworthy path between the smartcard and the server on the other end. In short, because everyone wants to be king of the mountain, nobody has been able to ascend.

Getting to a Secure Single Password

Reviewing our discussion, we can begin to see the form of a solution and can derive some requirements. Here, then, are mine:

- First, while a beautiful dream to some, a PKI deployed globally to all consumers has not happened yet, and there is no reason to believe conditions will change. Therefore, a solution should not rely on such a concept.
- No single vendor can own the market. Anyone playing King of the Mountain will be King of the Molehill. This implies use of open standards from the authenticator to the server, inclusively.
- The mechanism must be easy to use.
- The mechanism must handle multiple identities.
- The mechanism must be secure, not only from the network but from the host computer itself. This includes the computer keyboard!
- Finally, the mechanism must not cost an arm and a leg. We consumers are price-conscious!

With these requirements in mind, let's take a look at a straw man.

A Straw-Man Solution

What follows is not a complete answer to all consumer concerns. It is submitted for purposes of discussion.

Posit a smartcard with a small LCD display and a keypad (or other accessibility mechanisms) whose purpose is merely to provide mutual authentication for any single transaction. The interface between this card and a host computer would likely be USB, due to power concerns.

Each service will have its own identity, which will contain the following fields:

- A name that is a randomly generated 2048-bit number
- A nickname supplied by the service
- A 128-bit serial number
- A 2048-bit public/private key pair

The card will have the following functions:

- `longlonglong createIdentity(char *nickname, int *serial, char *public)`
- `confirmIdentity(longlonglong name, char *cryptext)`

These functions are called not by the host computer but by the remote server. They are only accessible *if and only if* the user confirms them on the smartcard itself.

Let us assume that the identity with the nickname of "Joe Blow's Bookstore" exists with a name of N and a serial number of S. When I connect to Joe Blow's Web site and want to authenticate a transaction, it will call the function `confirmIdentity()`. `cryptext` will be encrypted in the previously generated public/private key pair and will contain the next expected serial number and a comment indicating the nature of the transaction. Assuming that there is such a name on my smartcard and the `cryptext` is decrypted properly, if the serial number is correct, both it and the comment will be displayed on the smartcard, along with the nickname. If all looks correct to me, I push the green button. If it looks incorrect, I either do nothing or push the red button. Similarly, the card should report all failed access attempts.

What has this accomplished? First and foremost, no more passwords are sent on the wire—encrypted or not. Second, I no longer rely on the host computer for security. Third, I may store as many identities as I wish in as many smartcards as I wish. Fourth, no individual vendor can share an identity in such a way that a third party could make use of it for purposes of authentication without me knowing about it. Finally, no remote server will be able to guess even the mere existence of other identities on the card. Other authorized parties may tell them about them, but if they try to access the identity, I'll know something fishy is going on. Note that X.509 certificates are not needed for the common usage case.

What happens if someone other than Joe Blow's Bookstore tries to use the nickname for Joe Blow's bookstore? When the user is asked to create the new identity when he is not at Joe Blow's Bookstore, he should notice that something is wrong. However, even if he does create a new identity that has the same nickname, all this phisher has access to is its own fictitious identity and not the real Joe Blow's, because the names will differ.

The tricky part is in `createIdentity`. As with any authentication system, the weakest part is always in the bootstrapping process. Here the risk is that somehow the computer or other device between the card and the authentication server might eavesdrop or otherwise perform a man-in-the-middle attack. In order to protect against such a thing, that path must be encrypted. For this use only, a certificate may be warranted. However, one needs signed certificates for the server and for the card, not for the individual. The server just needs to verify that the endpoint is a sufficiently secure card.

Another approach would be to use out-of-band information, although we run the risk of having the same accessibility problems mentioned earlier. For instance, the user could enter the ID, as well as the serial number, directly on the card. This method is cumbersome to the user.

The astute observer will note that there is no method to list identities. Such a function should be considered dangerous, because if the authorized user can execute it from the host computer, then so might someone else. If such a listing is displayed, it should be displayed on the card itself.

Conclusion

There are numerous problems with the straw man above. The goal of the exercise was to provide some idea of what standards are needed and which *aren't* needed (X.509 for most transactions), and to demonstrate the sort of user interface that is required. An open standard is needed to exchange authentication information all the way from the card to the authentication server and back again.

Enterprises have an interest in this sort of solution as well, since solving the problem in the consumer space brings with it the consumer market's economies of scale. Tokens are already pretty cheap. Having to manage them, however, has been another matter entirely. That, too, could be addressed with such a solution.

The hardest part of this problem remains the registration of identities. In this limited sense, use of a PKI may be justified.

One limitation of my approach is that it doesn't easily allow for consolidation of credit cards, which are nothing more than keys themselves. Because identities are kept secret on the smartcard and are selected by the authorizing party, there is no way for the user to specify authorization of a charge on a particular account.

NOTES

1. S. Miller et al., "Kerberos: An Authentication Service for Open Network Systems," *Proceedings of the USENIX Winter Conference1988 (*February 1988), pp. 191–202.

2. C. Rigney et al., "Remote Authentication Dial-In User Service (RADIUS)," RFC 2865 (June 2000); W. Yeong et al., "X.500 Lightweight Directory Access Protocol," RFC 1487 (July 1993).

3. A. Frier et al., "The SSL 3.0 Protocol," Netscape Communications Corp. (November 18, 1996).

4. The SANS Institute, "Survival Time History" (August 2004): http://isc.incidents.org.

SANJAY GOEL AND STEPHEN F. BUSH

biological models of security for virus propagation in computer networks

Dr. Goel is an assistant professor in the School of Business and director of research at the Center for Information Forensics and Assurance at SUNY Albany. His research interests include distributed computing, computer security, risk analysis, biological modeling, and optimization algorithms.

goel@albany.edu
http://www.albany.edu/~goel

Dr. Bush is a researcher at GE Global Research. He continues to explore novel concepts in complexity and algorithmic information theory with a spectrum of applications ranging from network security and low-energy wireless ad hoc sensor networking to DNA sequence analysis for bioinformatics.

bushsf@research.ge.com
http://www.research.ge.com/~bushsf

THIS ARTICLE DISCUSSES THE SIMILARITY between the propagation of pathogens (viruses and worms) on computer networks and the proliferation of pathogens in cellular organisms (organisms with genetic material contained within a membrane-encased nucleus). It introduces several biological mechanisms which are used in these organisms to protect against such pathogens and presents security models for networked computers inspired by several biological paradigms, including genomics (RNA interference), proteomics (pathway mapping), and physiology (immune system). In addition, the study of epidemiological models for disease control can inspire methods for controlling the spread of pathogens across multiple nodes of a network. It also presents results based on the authors' research in immune system modeling.

The analogy between computers and communication networks and living organisms is an enticing paradigm that researchers have been exploring for some time. In 1984 Fred Cohen, in his Ph.D. dissertation, first put the term "computer virus" into print, although there he credits Len Adleman with coining the term used to describe the malicious pieces of code that can proliferate on a network and infect multiple computers. Since then, advances in bioinformatics (that is, the modeling of biological processes as well as storage, retrieval, and analysis of biological data through the use of information technology) have helped to define these analogies more precisely, to the point where results in bioinformatics can often be leveraged for use in computer networking and security. The challenges faced in bioinformatics are quite similar to those in computer network security. Several mechanisms have been devised in biological organisms to protect against pathogen invasion. It is important to learn from these biological phenomena and devise innovative solutions to protect computer systems from software pathogens.

Virus detection systems prevalent today are based on data analysis which looks for the presence of specific patterns. The data may be composed of header information in incoming packets at a firewall, data resident on a node, or behavioral patterns of programs resident on a computer. In most cases, the patterns of behavior (signatures) are defined a priori based on knowledge of existing pathogens. The signatures are usually gleaned from virus code by teams of virus experts who dissect the code and identify strings that uniquely identify the virus. The signature database in virus detection programs becomes obsolete rapidly, as new virus strains are released, and is updated as these new viruses are discovered. However, with the speed of virus propagation increasing—as is evident from the spread of the Slammer worm, which infected more than 90% of vulnerable hosts in 10 minutes—this mechanism is proving inadequate to control the spread of viruses, with its consequent loss of data and services. It is imperative to develop new virus detection software that does not rely solely on external intervention but can detect new strains of viruses by organically generating "antibodies" within a node. The physiology of cellular organisms contains several paradigms that can be used as inspiration for developing such autonomous security systems in computer networks. Several streams of research on automatic detection of virus (and worm) signatures are in progress (Kim and Karp, 2004), but this research is still preliminary and not mature enough for commercial deployment.

One of the initial areas explored in the realm of biological models of computer security involves the work of Forrest et al. (1994) with regard to virus detection. Here the similarities are strikingly clear regarding the need to quickly and efficiently identify viruses, generate "antibodies," and remove them from the system before they cause damage and propagate throughout the system. Prior to this, Kauffman (1969) had been focused on understanding and modeling the mechanics of gene transcription and translation within the cell. The concept of a complex network of interactions describing gene regulation had been born in the form of the Boolean network model. Now that the human genome has been fully sequenced, the task of determining gene function is a significant focus. However, specific genes identified in the sequence can interact with other genes in complex ways. Some portions of the genome can turn off the expression of other genes. These portions are called the structural and regulatory genes. Their behavior is thought to be a defense against foreign sequences, perhaps passed on from ancient viruses, from being expressed and potentially harming the organism (Hood, 2004). In fact, in this mechanism one can draw upon concepts that apply directly to network security, namely, the idea of defensive code that can be inherently activated to turn off dangerous code or viruses within the network. One of the problems in virus protection systems is the result of false positives, when portions of the code that provide legitimate functionality may be turned off accidentally. The authors propose use of surrogate code that can replicate the functionality of the pieces of code that are shut off, maintaining continuity in the operations of the node. Specifically, fault-tolerant networks are capable of surviving attacks and dynamically reconstituting services. Bush (2003) explores the ability of a communication network to genetically constitute a service. The network service evolves in real time using whatever building blocks are available within the network. Thus, a service damaged by a virus attack may be genetically reconstituted in real time. The general concept was illustrated using a specific example of a genetic jitter-control algorithm which evolved a 100-fold decrease in jitter in real time.

Another biological paradigm which lends itself well to adaptation as a computer security paradigm is protein pathway mapping. Living organisms have complex metabolic pathways consisting of interactions between proteins and enzymes, which may themselves have multiple subunits, alternate forms, and alternate specificities. Molecular biologists have spent decades investigating these biochemical pathways in organisms. These pathways usually relate to a known physiological process or phenotype and together constitute protein networks. These networks are very complex, with several alternate pathways through the same start and end point. The partitioning of networks into pathways is, however, often arbitrary, with the start and finish points chosen based on "important" or easily understood compounds. The models for biochemical pathways that have been developed thus far primarily demonstrate the working of the cellular machinery for specific tasks, such as metabolic flux and signaling. Several different modeling techniques are used: (1) classical biochemical pathways (e.g., glycolysis, TCA cycle); (2) stoichiometric modeling (e.g., flux balance analysis); and (3) kinetic modeling (e.g., CyberCell, E-Cell). More recently, cell metabolism is being studied using cellular networks that are defined from large-scale protein interaction and gene expression measurements.

Similar to the cellular networks in organisms, computer networks are complex in nature and collectively exhibit complex behavior. In these networks, start and end points can be arbitrarily chosen, and multiple paths may exist between the same nodes. Protein networks are predetermined and stay fairly static, whereas computer networks are constantly evolving with the addition of new nodes and network links. In protein networks, interactions among proteins, enzymes, and catalysts culminate in specific events. Analogously to protein networks, interactions among nodes of computer networks result in specific events or conditions in the network. The events may include propagation of viruses, denial-of-service attacks, and congestion on the network. Investigation of the network pathways along which the events propagate will enable us in forensic analysis to determine the root cause of the failures, as well as helping in developing intelligence for prediction of network events.

One biological paradigm that is not directly related to the physiology of living organisms is epidemiology that involves statistical analysis of disease propagation. Three basic models of disease propagation have been used extensively in epidemiological studies. Kephart and White (1991) first used these epidemiological models to study the spread of viruses on computer networks. Williamson and Léveillé (2003) have also developed virus spread models in computer networks using the epidemiological metaphor. Since then, several researchers have used variations of these basic models for studying the spread of computer viruses on computer networks.

The authors (Goel and Bush, 2003) have used the biological paradigm of the immune system, coupled with information theory, to create security models for network security. Information theory allows generic metrics and signatures to be created which transcend the specific details of a system or an individual piece of code. They compare information-theoretic approaches with traditional string-matching techniques. They also provide an analytic model that uses the epidemiological paradigm to study the behavior of the nodes. This article discusses several different biological paradigms which inspire defense against pathogens that invade computer networks, but it focuses on in-depth analysis of the immune system model. Some of the other innovative biological models that are currently being researched will be discussed in depth in a series of future articles.

Immune System Models

The role of the human immune system is to protect our body from pathogens such as viruses, bacteria, and microbes. The immune system consists of various kinds of cells, which operate autonomously and through interaction with each other to create complex chains of events leading to the destruction of pathogens. At a high level, cells can be categorized into two groups: detectors and effectors. Detectors identify pathogens, and effectors neutralize them. There are two kinds of immune responses evoked by the immune system: innate response and adaptive response. The innate immune response is the natural resistance of the body to foreign antigens and is non-specific toward invaders in the body. During this response, a specialized class of cells called phagocytes (macrophages and neutrophils) is used. These specialized cells, which have surface receptors that match many common bacteria, have remained unchanged throughout evolution. This system reacts nearly instantaneously to detect pathogens in the body. However, it is incapable of recognizing viruses and bacteria that mutate and evolve.

The innate immune response is complemented by the adaptive immune response, in which antibodies are generated to specific pathogens that are not recognized by the phagocytes. The adaptive response system uses lymphocytes, which have receptors for a specific strain instead of having receptors for multiple strains as phagocytes do. Lymphocytes are produced in the bone marrow, which generates variants of genes that encode the receptor molecules and mature in the thymus. When an antigen is encountered, it is presented to the lymphocytes in the lymphatic system. The lymphocytes that match proliferate by cloning and subsequently differentiate into B-cells, which generate antibodies, and T-cells, which destroy infected cells and activate other cells in the immune system. Most effectors that proliferate to fight pathogens die; only 5–10% are converted into memory cells which retain the signature of the pathogen that was matched. These memory cells permit a rapid response the next time a similar pathogen is encountered, which is the principle used in vaccinations and inoculations. The number of memory cells produced is directly related to the number of effector cells in the initial response to a disease. While the total number of memory cells can become quite large, still, as an organism is exposed to new pathogens, newer memory cells may take the place of older memory cells, due to competition for space (Ahmed, 1998). This decrease in memory cells leads to weakened immunity over time. Another reason for weakened immunity is an immune response rate that is not sufficiently rapid to counteract the spread of a powerful exotoxin, such as that produced by tetanus (Harcourt et al., 2004). Lymphocytes have a fixed lifetime, and if during this period they do not match a pathogen, they automatically die.

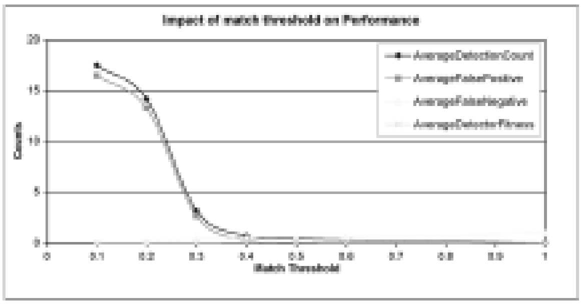
The key to the functioning of the immune system is detection. Recognition is based on pattern matching between complementary protein structures of the antigen and the detector. The primary purpose of the genetic mechanism in the thymus and bone marrow is to generate proteins with different physical structures. The immune system recognizes pathogens by matching the protein structure of the pathogen with that of the receptor. If the receptor of the antigen and the detector fit together like a three-dimensional jigsaw puzzle, a match is found. A fundamental problem with the detection mechanism of the immune system is its computational complexity. For example, if there are 50 different attributes with four different values, over six million different detectors are required to cover the search space. The number of virus structures that can arise due to different protein configurations is virtually infinite. In spite of high efficiency in creating detectors and pattern matching at the molecular level, maintaining a detector for each possible pathogen protein structure is not feasible. The human immune mechanism solves this problem by using generalizations in matching—that is, some features of the structure are ignored during matching. This is called specificity of match; the more features are ignored, the lower the specificity. The lower the specificity, the fewer the number of detectors required for matching a population of pathogens and the more nonspecific is the response. An explanation of specificity is elegantly described in J.H. Holland's description of classifier systems (1985). To cover the space of all possible nonself proteins, the immune system uses detectors with low specificity. This enables the immune system to detect most pathogens with only a few detectors; however, it results in poor discrimination ability and a weak response to pathogen intrusion. The immune system counters this problem by employing a process called affinity maturation (Bradley and Tyrrell, 2000). Several methods have been proposed for analytic representation of matching pathogen signatures in the immune system, such as bit-strings (Farmer, Packard, and Perelson, 1986; De Boer, Segel, and Perelson, 1992), Euclidean parameter spaces (Segel and

Perelson, 1988), polyhedron models (Weinand, 1991), and, more recently, Kolmogorov Complexity (Bush, 2002; Goel and Bush, 2003).

Several applications based on immune systems outside the area of biology have recently emerged, the most notable of these being computer security. Kephart (1995) was perhaps the first to introduce the idea of using biologically inspired defenses against computer viruses and immune systems for computer security. Forrest et al. (1994) also proposed the use of immune system concepts for design of computer security systems and provided an elaborate description of some immune system principles applicable to security. They presented three alternate matching schemes—Hamming distance, edit distance, and r-contiguous bits—arguing that the primary premise behind a computer immune system should be the ability to distinguish between self and non-self. They presented a signature scheme where a data tuple consisting of source IP address, destination IP address, and a destination port number were used to distinguish self-packets from non-self packets. Hofmeyr (1999) presented a detailed architecture of a computer immune system. He analytically compared different schemes for detection of pathogens, such as Hamming distance and specificity. There are several other works in the literature on the use of immune systems for network security, including Murray (1998), Kim and Bentley (1999), and Skormin et al. (2001). Kephart and White (1991, 1993) present an architecture for an immune system and the issues involved in its commercialization. They incorporate a virus analysis center to which viruses are presented for analysis through an active network. The Kolmogorov Complexity approach (Goel and Bush, 2003) demonstrated a 32% decrease in the time required to detect a signature over two common Hamming distance–based matching techniques, i.e., a sliding window and the number of contiguous bit matches. The Kolmogorov Complexity–based technique estimates the information distance of entire code sequences, not just specific segments or bits. Using the entire code sequence makes it more difficult to modify the virus so that it can hide in another portion of a legitimate code segment.

Artificial immune systems consist of detectors and effectors that are able to recognize specific pathogen signatures and neutralize the pathogens. To detect pathogens, the signature of incoming traffic packets is matched against signatures of potential viruses stored in an immune system database. An immune system that is capable of recognizing most pathogens requires a large number of detectors. Low-specificity detectors that identify and match several viruses are often used to reduce the number of detectors at the cost of increased false positives. The computational complexity of a computer immune system remains fairly high, and individual nodes are incapable of garnering enough resources to match against a large signature set. The computational complexity gets worse as network traffic grows due to use of broadband networks, and it is straining the capacities of conventional security tools such as packet-filtering firewalls. Massive parallelism and molecular-level pattern matching allow the biological immune system to maintain a large number of detectors and efficiently match pathogens. However, artificial immune systems have not achieved these levels of efficiency. To reduce the computational burden on any individual node in the network, all nodes need to pool their resources, share information, and collectively defend the network. In addition, such inspection should be done within the network itself, to improve efficiency and reduce the time required for reacting to an event in the network. This concept of collective defense enabled by a unified framework is the primary premise of the authors' research. To enable this concept of collective network defense, they have proposed an approach based on information theory principles using Kolmogorov Complexity measures.

To study the parameters and different schemes of detection and sampling in the immune system, Goel et al. (working paper) have developed a simulation model using RePast (Schaeffer et al., 2004), a simulation tool typically used for modeling self-organizing systems. The simulation models a classical immune system, where new signatures are created by mutation of existing signatures which then go through a maturation phase. The simulation also models a cooperative immune system, where multiple nodes on the network share virus detection information prevalent in the network to improve the efficiency of each immune system. The research will investigate the trade-off between the additional burden of sharing information across nodes and the benefit of improving scanning efficiency by obtaining intelligence information on active or new pathogens. Figures 1a and 1b show the impact of the match threshold and sampling rate, respectively, on the performance of the immune system. Figure 1a shows a high gradient between a threshold match of 0.2 and 0.4, which is the practical operating region for the immune system. Figure 1b shows an improved performance with the sampling rate, which asymptotes around 70%.



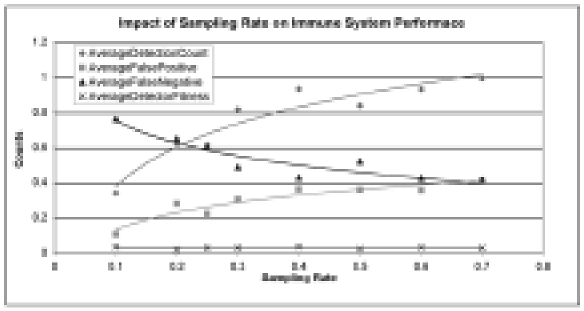


Figure 1. Plots showing impact of match threshold and sampling rate on immune system-metrics

Goel and Bush (2003) have also compared different signature metrics and have demonstrated that Kolmogorov Complexity is a feasible metric for the signature of pathogens.

Conclusion

The security models for detection and elimination of pathogens that invade computer networks have been based on perimeter defense. Such defenses are proving inept against fast-spreading viruses and worms. The current tools are unable to guarantee adequate protection of data and unfettered access to services. It is imperative to complement these existing security models with reactive systems that are able to detect new strains of pathogens reliably and are able to destroy them before they can cause damage and propagate further. Several biological paradigms provide a rich substrate to conceptualize and build computer security models that are reactive in nature. Three specific mechanisms in mammalian organisms present the most potential: (1) the RNAi mechanism, (2) protein pathway mapping, and (3) the immune mechanism. In addition, the models of disease control that study the spread and control of viruses suggest ways to throttle the spread of viruses. Current work has mainly focused on the use of immune and epidemiological models. It is time to move beyond these existing models to other innovative models, such as those based on genomics and proteomics. Such reactive models provide a scalable, resilient, and cost-effective mechanism that may keep pace with constantly evolving security needs.

Acknowledgments

The authors are grateful to Damira Pon, from the School of Information Science at the State University of New York at Albany, for a thorough review of this article and her useful suggestions.

REFERENCES

Ahmed, R., February 5, 1998. “Long-term Immune Memory Holds Clues to Vaccine Development,” Emory Health Sciences Press Release.

Bradley, D.W., and Tyrrell, A.M., April 2000. “Immunotronics: Hardware Fault Tolerance Inspired by the Immune System,” *ICES 2000* (Springer-Verlag, 2000), pp. 11–20.

Bush, S.F., 2002. “Active Virtual Network Management Prediction: Complexity as a Framework for Prediction, Optimization, and Assurance,” *Proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE 2002)* (Los Alamos, CA: IEEE Computer Society Press), pp. 534–553: http://www.research.ge.com/~bushsf/ftn/005-FINAL.pdf.

Bush, S.F., 2003. “Genetically Induced Communication Network Fault Tolerance,” *Complexity Journal*, vol. 9, no. 2, Special Issue: “Resilient & Adaptive Defense of Computing Networks”: http://www.research.ge.com/~bushsf/pdfpapers/ComplexityJournal.pdf.

Cohen, F., 1987. “Computer Viruses Theory and Experiments,” *Computers and Security*, vol. 6, pp. 22–35.

De Boer, R.J., Segel, L.A., and Perelson, A.S., 1992. “Pattern Formation in One- and Two-Dimensional Shape-Space Models of the Immune System,” *J. Theor. Biol.*, pp. 155, 295–333.

Farmer, J.D., Packard, N.H., and Perelson, A.S., 1986. “The Immune System, Adaptation, and Machine Learning,” *Physica D*, vol. 22, pp. 187–204.

Forrest, S., Perelson, A.S., Allen, L., and Cherukuri, R., 1994. “Self–Nonself Discrimination in a Computer,” *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy* (Los Alamos, CA: IEEE Computer Society Press).

Goel, S., and Bush, S.F., 2003. “Kolmogorov Complexity Estimates for Detection of Viruses in Biologically Inspired Security Systems: A Comparison with Traditional Approaches,” *Complexity*, vol. 9, no. 2: http://www.research.ge.com/~bushsf/pdfpapers/ImmunoComplexity.pdf.

Goel, S., Rangan, P., Lessner, L., and Bush, S.F., [working paper]. “Collective Network Defense: A Network Security Paradigm Using Immunological Models.”

Harcourt, G.C., Lucas, M., Sheridan, I., Barnes, E., Phillips, R., Klenerman, P., July 2004. “Longitudinal Mapping of Protective CD4 T Cell Responses Against HCV: Analysis of Fluctuating Dominant and Subdominant HLA-DR11 Restricted Epitopes,” *Journal of Viral Hepatitis*, vol. 11, no. 4, p. 324.

Hofmeyr, S.A., May 1999. “An Immunological Model of Distributed Detection and Its Application to Computer Security,” Ph.D. thesis, University of New Mexico.

Holland, J.H., 1985. “Properties of the Bucket Brigade Algorithm,” *Proceedings of the 1st international Conference on Genetic Algorithms and Their Applications*, ed. Grefenstette, J.J., L.E. Associates, pp. 1–7.

Hood, E., 2004. “RNAi: What’s All the Noise About Gene Silencing?” *Environmental Health Perspectives*, vol. 112, no. 4.

Kauffman, S.A., 1969. “Metabolic Stability and Epigenesis in Randomly Constructed Genetic Nets,” *J. Theor. Biol.*, vol. 22, pp. 437–467.

Kephart, J. O., 1995. “Biologically Inspired Defenses Against Computer Viruses,” *Proceedings of IJCA ’95*, pp. 985–996.

Kephart, J.O., and White, S.R., 1991. “Directed Graph Epidemiological Models of Computer Viruses,” *Proceedings of the 1991 IEEE Computer Security Symposium on Research in Security and Privacy*, pp. 343–359.

Kephart, J.O., and White, S.R., May 1993. “Measuring and Modeling Computer Virus Prevalence,” *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy,* pp. 2–15.

Kim, H.-A., and Karp, B. 2004. “Autograph: Toward Automated, Distributed Worm Signature Detection,” *Proceedings of the 13th USENIX Security Symposium,* pp. 271–286.

Kim, J., and Bentley, P., 1999. “Negative Selection and Niching by an Artificial Immune System for Network Intrusion Detection,” *Late-Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference (GECCO '99),*, pp.149-158.

Murray, W.H., 1998. “The Application of Epidemiology to Computer Viruses,” *Computer Security*, vol. 7, pp. 139–150.

Schaeffer, S.E., Clemens, J.P., and Hamilton, P., 2004. “Decision Making in a Distributed Sensor Network,” *Proceedings of the Santa Fe Institute Complex Systems Summer School:* http://www.tcs.hut.fi/~satu/online-papers/sensor.pdf.

Segel, L.A., and Perelson, A.S., 1988. “Computations in Shape Space: A New Approach to Immune Network Theory,” in *Theoretical Immunology Part 2*, ed. Perelson, A.S. (Redwood City: Addison-Wesle)y, pp. 377–401.

Skormin, V.A., Delgado-Frias, J.G., McGee, D.L., Giordano, J.V., Popyack, L.J., Gorodetski, V.I., and Tarakanov, A.O., 2001. “BASIS: A Biological Approach to System Information Security,” *Mathematical Methods, Models, and Architectures for Network Security Systems (MMM-ACNS) 2001*, pp. 127–142.

Weinand, R.G., 1991. “Somatic Mutation and the Antibody Repertoire: A Computational Model of Shape-Space,” in *Molecular Evolution on Rugged Landscapes*, ed. Perelson, A.S. and Kaufman, S.A., *SFT Studies in the Science of Complexity*, vol. 9 (Redwood City: Addison-Wesley), pp. 215–236.

Williamson, M.M., and Léveillé, J., 2003. “An Epidemiological Model of Virus Spread and Cleanup”: http://www.hpl.hp.com/techreports/2003/HPL-2003-39.html.

JONATHAN S. SHAPIRO

extracting the lessons of Multics

Jonathan Shapiro is an assistant professor at Johns Hopkins University. His research interests focus on operating systems and system security. He is also a recidivist entrepreneur and has seen the development of several products from technical concept to market.

shap@eros-os.org
http://www.eros-os.org/~shap

THE KEYNOTE SPEAKER FOR THIS year's USENIX Security Conference was Earl Boebert, a key participant in the Multics project. Interspersed with pointed (and painfully accurate) insights about the state of the computing field, Dr. Boebert's talk focused on the structure of the Multics system, its key features, how many security problems it didn't have, and how little has been learned from it in the intervening 40 years. Karger and Schell made much the same claim in their retrospective paper.[1] In spite of being one of the best documented early operating system projects in existence,[2] and a staggeringly innovative effort in its own right, Multics is primarily remembered today as the system whose cost and ongoing delays caused Bell Labs to withdraw from the Multics collaboration and ultimately led Thompson, Ritchie, McIlroy, and Ossana to start the UNIX project. It seems fitting to try to answer the question Boebert posed in his talk: What can we learn from the past? Why did Multics fail?

The Multics project was proposed in 1962 by J.C.R. Licklider (the founder of DARPA) as part of his commitment to connected, multi-user computing. It was a year of dramatic invention. The notion of computer-supported collaboration, and therefore multi-user, time-shared computing in some form, was definitely "in the ether." Doug Engelbart and Ted Nelson would independently invent hypertext in 1962.[3] Berkeley, Dartmouth, and several other groups were exploring time-sharing systems, and 1962 was also the year that brought us Spacewar, the first computer game. The mouse would come shortly as Engelbart gained experience with Augment and, later, NLS (early hypertext systems). In the middle of this, a bunch of technologists decided to invent the modern computing utility. The Multics contract was awarded by DARPA in August 1964.

While few of the major innovations embodied in the Multics system were original to Multics, it was probably the first attempt to integrate so many ideas effectively. Virtual memory, segment-based protection, a hierarchical file system, shared memory multiprocessing, security, and online reconfiguration were all incorporated into the Multics design. Multics may have been the first system implemented primarily in a high-level programming language, and it was one of the first to support multiple programming languages for creating applications. MACLISP, troff, and many other early tools trace their origins to the Multics system, as does much of the modern UNIX command line. Multics originated the term “computer utility,” a concept that we have yet to fully explore 40 years later The integrated circuit was patented in 1959 and by 1963 was just entering the scene in the form of the 7400-series logic parts. Volume customers might soon expect to get as many as four *gates* on a single chip. . Considered in the context of then-available electronics technology, Multics was an incredible undertaking.

Today, we take for granted (at least, we *say* we do) many of the software techniques pioneered in the Multics effort. Multics was one of the first attempts at serious software engineering in a large, general-purpose system. It established the use of small, enforcedly encapsulated (isolated) software components (subsystems) that could be invoked only through their published interfaces. Using this fundamental building block, the Multicians crafted an end-to-end design that was both robust and secure. Even a casual reading of the Orange Book (TCSEC) standard reveals that many of the ideas of the Orange Book originated in either the Multics architecture or the Multics software process.

A skeptic might be prompted to ask, “If Multics was so wonderful, why aren't we using it today?” It is tempting to think, as Boebert implies, that Multics was a victim of the American desire for “crap in a hurry,” and that this did not allow for the emphasis on quality engineering that delayed the completion of Multics. The truth, I suspect, is a matter of economics rather than bad taste. Multics was largely doomed by the intersection of two forces: the exploding growth of the computer and semiconductor industries, and a rising national sensibility of individual empowerment that brought, inevitably, the trend toward decentralized computing.

Exponential advances in integrated circuit design conspired against the Multics effort. In 1962, Fairchild Semiconductor was still shipping individual transistors, and this was “state of the art.” By the time Bell Labs withdrew from the Multics project in 1969, Intel was shipping 64-bit memory chips. By the time Multics was presented commercially in 1973, Digital Equipment Corporation was shipping entry-level versions of PDP-11 systems running either RSTS-11 or RSX-11. DEC would ship the LSI-11 and the PDP-11/70 two years later, and in doing so would establish the features that would ultimately define mid-range computer architecture. In the microprocessor world, the 8080 would be running early versions of CP/M by 1974. A new era had arrived.

When it was announced in 1973, Multics was arguably the perfect answer to the problems of 1964, but it was too late, too expensive, too dependent on a proprietary hardware architecture, and too focused on centrally shared computing to be relevant in a world where decentralized, departmental computing was becoming the order of the day. Following the pattern of every other competitive market in history, the mainframe was being commoditized from below, and the era of “personal computing” would soon take over the world. Key elements of Multics—virtual memory, the hierarchical file system, multiple user support, and, to a lesser degree, online reconfiguration—would be rediscovered and incrementally introduced on Digital's VAX line of hardware, burdened at each step with the requirements of backwards compatibility. The same process of reinvention is happening now as Microsoft reshapes the underlying PC standard to support advanced server and management features. Not until the arrival of universal, always-on connectivity would the lessons of Multics once again seem relevant.

Ultimately, Multics failed because high-end computer-architecture ideas consolidated around the VAX and System 360 feature set, and shifted away from features such as segmentation and multiple protection rings, on which Multics relied. This left the Multics operating system nonportable and unable to exploit the shift to cheaper, more open hardware systems. Unfortunately, it occurred at a time when circuit integration still wasn't far enough along to support basic protection features on low-end microprocessors. A decade later, UNIX would benefit from being able to ride a more stable, mature technology curve, reaping benefit from the same forces that doomed the Multics effort.

Sadly, later systems would show that neither segmentation nor multilevel protection rings were necessary. A simple user/supervisor split coupled with a paged memory management unit is sufficient. The Gemini project would construct an A1-certifiable operating system on a 386-class microprocessor. KeyKOS would provide Multics-comparable levels of robustness and security on commodity microprocessors. The EROS research effort has directly adopted many of these ideas. They seem to be contagious—the L4 project is now adopting many of them as well.[4] The "lots of small, protected memory objects" approach of Multics is reframed in KeyKOS, EROS, and successors as a "lots of small, protected processes" approach, which has stood up well to both formal and practical testing.

Unfortunately, neither UNIX nor Microsoft Windows managed to preserve (or successfully replace) the key security underpinnings of Multics, and we are now committed to a large body of insecure legacy software that will be difficult to overcome at a time when software patents make overcoming an entrenched legacy provider nearly impossible.

Given the economic and technology environment into which it emerged, the wonder of Multics is not that it has been ignored, but that so many of its key ideas have been adopted and adapted so pervasively in later efforts. Multics largely defined modern time-sharing systems, and its influence can be seen in every multi-user system that is shipping today.

FURTHER READING

1. Paul A. Karger and Roger R. Schell, "Thirty Years Later: Lessons from the Multics Security Evaluation." *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)* (December 2002)

2. Elliot I. Oganick, *The Multics System: An Examination of Its Structure* (Cambridge, MA: MIT Press, 1972). The best starting point today for information about the Multics system is the Multicians' Web site at http://www.multicians.org.

3. D.C. Engelbart, "Augmenting Human Intellect: A Conceptual Framework" (Stanford Research Institute, 1962); see Theodor H. Nelson, *Literary Machines 931* (Mindful Press, 1982)

4. KeyKOS: http://www.cis.upenn.edu/~KeyKOS; EROS: http://www.eros-os.org.
L4: http://www.l4ka.org.

PETER H. SALUS

the bookworm

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

peter@netpedant.com

BOOKS REVIEWED IN THIS COLUMN

MOVING TO THE LINUX BUSINESS DESKTOP
Marcel Gagne
Boston, MA: Addison-Wesley, 2004. Pp. 638 + CD-ROM. ISBN 0131421921.

LINUX COOKBOOK
Michael Stutz
2nd ed., San Francisco: No Starch, 2004. Pp. xxxi + 788. ISBN 1593270313.

HIGH-TECH CRIMES REVEALED
Steven Weber
Cambridge, MA: Harvard U.P., 2004. Pp. 312. ISBN 0-674-01292-5.

SUCCEEDING WITH OPEN SOURCE
Steven Branigan
Boston, MA: Addison-Wesley, 2004. Pp. xxix + 412. ISBN 0321218736.

BUDGETING FOR SYSADMINS
Adam Moskowitz
Short Topics in System Administration #10, Berkeley, CA: USENIX, 2003. Pp. 37. ISBN 1-931971-12-9.

THE TURING TEST
Stuart Shieber, ed.
Cambridge, MA: MIT Press, 2004. Pp. 346. ISBN 0262692937.

Time for the December column again! And I gotta pick my annual holiday "best" list. Let me say here that it was no easy pick. Last year, Gibson, Stephenson, and Waldrop weren't eligible, so that gets rid of Sterling this year. But there is a consolation: I also get to list a bonus book. But first, the books at hand.

TWO PENGUINS

Gagne's keyboard must generate a lot of heat. Earlier this year there was *Moving to Linux*, and now there's a large *Business Desktop* version. While there is a lot of space in this "new" book given over to material from the earlier one, there is an admirable amount of new material. I think Gagne does a really fine job. But I'm afraid that this isn't a book for a true newbie, either. Gagne explains Samba very well, and I was thrilled by his exposition on OpenOffice, but this is not a book that can be used readily by someone with no computer background. If you've got some command line experience, this will be the perfect book. It comes with a Knoppix CD.

Stutz' *Linux Cookbook* has waxed since 2000, when the first edition appeared, from 402 to over 800 pages. But there's a lot more useful stuff here. Well worth getting.

BLACK HATS

Branigan has been involved in high-tech forensics at Bell Labs and at Lumeta for quite a while. With *High-Tech Crimes Revealed* he's produced something quite out of the ordinary: a "security" book as compelling as a whodunit. Peppering his book with anonymized anecdotes, he's given us a gripping story.

MONEY PROBLEMS

When I was in Boston at the ATC, I saw Moskowitz' pamphlet. For some reason, SAGE hadn't sent it to me, so this brief review is late. I apologize, mainly because this is 30 pages that should be read and re-read by anyone who needs to construct and present a budget, whether large or small. Adam, it's a very fine job.

TURING COMPLETE

I would guess that 60% of the books I receive each year contain the name "Turing" at some point. Usually in the phrases "Turing complete" or "Turing test." I would guess further that over half of those uses are irrelevant (stuck in for intellectual color) or just inappropriate. Shieber has put together a truly superb anthology concerning the "indistinguishability test" (between artifact and person).

Beginning with Descartes and La Mettrie and moving through all four of Turing's papers, Shieber provides us with 13 essays for and against the "test," ending with Chomsky's "Turing on the 'Imitation Game'" (written for this anthology in 2002). A great read.

THE VERY BEST

Marshall Kirk McKusick and George V. Neville-Neil, *The Design and Implementation of the FreeBSD Operating System*, Addison-Wesley.

Lawrence Rosen, *Open Source Licensing*, Pearson Education.

Steven Weber, *The Success of Open Source*, Harvard U.P.

R. Kent Dybvig, *The Scheme Programming Language*, MIT Press.

Craig Hunt, *Sendmail Cookbook*, O'Reilly.

Cyrus Peikari & Anton Chuvakin, *Security Warrior*, O'Reilly.

Diomidis Spinellis, *Code Reading*, Addison-Wesley.

Arnold Robbins, *Linux Programming by Example*, Prentice Hall.

Mark Burgess, *Analytical Network and System Administration*, John Wiley.

Jonathan Land, *The Spam Letters*, No Starch.

Last year's "bonus" was the then-new *UserFriendly* book. This year it goes to cartoons again: Nitrozac and Snaggy, *The Best of the Joy of Tech*, O'Reilly.

RIK FARROW

book reviews

I guess I get a lot of books for review because I began writing articles about UNIX and security back in 1986. I get more books than I can read, as well as some I don't think I would ever read—ones whose topics are far afield from what I really need to know about.

Recently, I received several books all with related topics, and thought that they deserved special treatment. Two of the books are about using Snort, while the third does mention Snort and IDS but represents a much deeper topic: network security monitoring.

MANAGING SECURITY WITH SNORT AND IDS TOOLS

Kerry Cox and Christopher Gerg

Sebastopol, CA: O'Reilly and Associates, 2004. Pp. 269.
ISBN 0-596-00661-6.

SNORT 2.1 INTRUSION DETECTION, 2D ED.

Jay Beale et al.

Rockland, MA: Syngress Publishing Inc., 2004. Pp. 716.
ISBN 1-931836-04-3.

Jay Beale is the editor of the *Snort 2.1* book, as there are actually many authors. The cast of characters involved in *Snort 2.1* is both an advantage and a disadvantage. On the plus side, you get chapters written by Snort developers. On the minus side, you get a book that could be better organized and that still contains some mistakes and typos which also plagued the first edition.

Managing Security with Snort is shorter and written by Snort users rather than developers and users. Being an O'Reilly book, it is formatted differently and, as a result, is better organized than the Syngress book. I found the instructions for building and using Snort, or a Snort-related tool like ACID, clear and easy to follow (ACID has been and still is a bear to build).

There are certainly differences deeper than formatting between these books. While either will get you going with Snort, *Snort 2.1*, with its greater length, does get into more details. For example, *Managing Security* has five pages on configuring and using Barnyard, a tool for processing Snort alerts asynchronously. *Snort 2.1* devotes an entire chapter, 75 pages, to working with Barnyard.

I had wondered how these books would handle Snort rule writing. The two deal with this topic in almost the same manner. Each book explains the parts of Snort rules and provides a couple of examples, but neither one has a tutorial. I consider rule writing/editing a critical topic in a rule-based IDS tool, and was disappointed that neither book goes deeply enough into this area. *Managing Security* actually misses an important new set of rule options, `flow`, which allows rules to include the distinction of a packet going to a client or a server.

I can recommend either of these books, and suggest that you make your decision based on how deeply you want to go into Snort and related tools.

THE TAO OF NETWORK SECURITY MONITORING

Richard Bejtlich

Boston: Addison-Wesley, 2004. Pp. 798.
ISBN 0-321-24677-2.

Bejtlich honed his network monitoring skills working for the Air Force Computer Emergency Response Team, and his experience shows. Instead of talking about preventing intrusions, Bejtlich assumes there will be intrusions. His approach involves collecting as much network data as possible, so this information can be used to determine what has happened on your network in the past. The data includes IDS alerts, flow data, and complete packet dumps.

Bejtlich successfully explains why IDS alerts are not enough. He demonstrates how having both flows and packet data helps analysts determine when an alert represents a successful attack. Bejtlich does briefly mention Snort, but he covers many other tools as well. Bejtlich is particularly fond of Sguil, a tool for querying back-end databases that contain the mountain of alerts, flows, and packets generated in his approach.

Bejtlich presents network and system management (NSM) as a philosophy, backed with lots of practical advice from someone who uses NSM in real life. I highly recommend this book if you are serious about your network security and want to go beyond viewing IDS alerts.

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, Tcl, Perl, Java, and operating systems, book reviews, and summaries of sessions at USENIX conferences.

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

ACCESS TO PAPERS from USENIX conferences online, starting with 1993: 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

USACO: The International Finals

by Rob Kolstad

The 2004 International Olympiad on Informatics (the final leg of the programming contest circuit for pre-college programmers) was held in Athens, Greece, September 11–18. Athens was still in "Olympic mode": The 700 IOI participants stayed in one of the buildings erected for Olympic journalists; the Paralympics overlapped the IOI by about five days.

Athens is a bustling place full of new Olympic construction, fine Mediterranean weather, and a "can do" attitude.

I arrived several days early to set up and tune the automated grading system. I ended up spending six days in the beautiful basement/parking garage where the actual competitions were held. The parking garage was chosen for the actual contest because the addition of a few walls, power, lights, and air-conditioning made it workable, and it was one of the few places in the vicinity that had enough room for 300 PCs, desks, chairs, aisles, and networking equipment.

The USA delegation, which included not only our four finalists, Brian Jacokes, Anders Kaseorg, Eric Price, and Alex Schwendner, but also Don Piele (USACO director and this year's chair of the main IOI governing body), Greg Galperin (the USA delegate to the International Scientific Committee), coaches Hal Burch and Brian Dean, and some spouses/visitors.

Contestants were challenged with two five-hour contests, each with three tasks. These tasks are generally very difficult and concentrate on algorithms (rather than, say, systems programming, administration, or databases). Here—written by coach Brian Dean—is a typical problem from the USACO March 2004 competition:

Moo University Gymnastics Team Tryouts

N (1 <= N <= 1,000) calves try out for the Moo U gymnastics team this year, each with a positive integer height and a weight less than 100,000. Your goal is to select a team of as many calves as possible from this group. There is only one constraint the team must satisfy: The height H and weight W of each calf on the team must obey the following inequality:

A*(H–h) + B*(W–w) <= C

where h and w are the minimum height and weight values over all calves on the team, and A, B, and C are supplied positive integral constants less than 10,000. Compute the maximum number of calves on the team.

The USA tied their best performance ever, with two gold medals. Brian Jacokes placed 6th overall; Anders placed 13th. Eric just missed a gold medal by 10 points out of 600; Alex was only 25 points out. Both Eric and Alex have one more year of eligibility.

Over the (North American) 2004–2005 school year, USACO will hold half a dozen Internet-based contests for pre-college students before the USA Invitational Olympiad next June; the contests are open to all pre-college students on the Internet. No entry fees are charged. Teachers and students can sign up for the low-traffic mailing list by sending a "subscribe hs-computing" email to hs-computing@usaco.org.

We're working hard this year to expand the competition levels so that students of all abilities can compete in C, C++, Pascal, and Java. Please tell those who might benefit!

USENIX is a major sponsor of the USA Computing Olympiad. Other sponsors include SANS, the ACM, IBM, and Google. If your organization would like to contribute (in any way—e.g., we need a few higher speed servers for running contests), please contact me.

Thanks to Our Volunteers

by Ellie Young
USENIX Executive Director

USENIX's success would not be possible without the volunteers who lend the 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 2004:

The program chairs for our 2004 conferences:

Chandu Thekkath, Third Conference on File & Storage Technologies

Robert Morris and Stefan Savage, First Symposium on Networked Systems Design & Implementation

Tarek S. Abdelrahman, Third Virtual Machine Research & Technology Symposium

Andrea Arpaci-Dusseau and Remzi Arpaci-Dusseau, 2004 USENIX Annual Technical Conference

Matt Blaze, 13th USENIX Security Symposium

Ted Ts'o for organizing the 2004 Linux Kernel Developers Summit

Lee Damon, 18th LISA Conference

Eric Brewer and Peter Chen, 6th Symposium on Operating Systems Design & Implementation

David Culler and Timothy Roscoe, 1st Workshop on Real, Large Distributed Systems

The conferences' Invited Talk/Special Track chairs:

For USENIX '04 Annual Tech:

Bart Massey and Keith Packard, 2004 Freenix Program Chairs at USENIX Annual Tech

Murray Stokley, Avi Rubin, Ted Ts'o, Rob Kolstad, and Jon "maddog" Hall for serving as SIG Session chairs

Peter Salus for lining up the plenary speakers

Clem Cole for organizing the Guru Is In sessions

Esther Filderman for organizing the AFS workshop

For the 13th USENIX Security Symposium:

Avi Rubin and Vern Paxson

For LISA '04:

Deeann Mikula, Adam Moskowitz, and Marcus Ranum for the invited talks

Phil Kizer for the Guru Is In sessions

Gretchen Phillips for the workshops

Other major contributors:

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

Victor Bahl for his efforts as liaison and steering committee chair for the SIGMOBILE/USENIX MobiSys conference

Peter Honeyman for his eight years of service on the USENIX Board of Directors (1996–2004) and for his continued efforts in reaching out to other groups, international and domestic: e.g., OpenAFS community, SANE conference, Smartcards/CARDIS, and Middleware conference

John Gilmore, Avi Rubin, Lois Bennett, and Tina Darmohray for their 12 years of service on the USENIX Board

Mike Jones, Clem Cole, Alva Couch, Ted Ts'o, Jon Hall, Kirk McKusick, Geoff Halprin, and Matt Blaze for their service on the USENIX Board in 2004

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

Mike Jones for serving as liaison to the Computing Research Association

USENIX is grateful to all!

conference reports

- This issue's reports focus on the 13th USENIX Security Symposium, held in San Diego, California, August 9–13, 2004.
- Our thanks to the scribe coordinator:

  Rik Farrow
- Our thanks to the summarizers:

  Alvin AuYoung

  Eric Cronin

  Marc Dougherty

  Serge Egelman

  Rachel Greenstadt

  Stefan Kelm

  Zhenkai Liang

  Chad Mano

  Nick Smith

  Ashish Raniwala

  Tara Whalen

  Wei Xu

KEYNOTE ADDRESS

Back to the Future

William "Earl" Boebert, Sandia National Laboratory

Summarized by Tara Whalen

Earl Boebert opened his remarks by saying that his views were his own, since "nobody in their right mind would let me speak for them." The wealth of knowledge and insight expressed in his keynote talk demonstrated that nobody in their right mind would ignore his expertise. Boebert stated that a lot has been forgotten about security over the years, so it is necessary for somebody who's been around for a while to speak up.

Boebert said that the way to build buildings that stay up is to look at buildings that fall down. Security experts should not ask, "Why does it work?" but "How does it fail?" What is worrisome is that currently insecurity has been pushed to the end nodes, which is the worst place for it to be. Why did this happen? It was driven by economics: You make money by giving people what they want. Quoting comedian Rich Hall, Boebert said that what Americans wanted was "crap in a hurry," and, apparently, so do computer people. However, an alternative exists: engineering. Software developers used to aspire to well-engineered solutions before the industry was overwhelmed with the "get rich quick" ethos.

Boebert went on to talk about the higher-level goals of engineering: operational and formal assurance. Operational assurance is what you gain from experience: "It hasn't killed anybody yet, so it must be okay." However, to gain operational assurance, you must always carry out your operations in exactly the same way. Instead, you could rely on formal assurance, a structured argument that shows what you can safely do. Boebert then talked about assurance in the context of his experience with Multics.

Multics was developed around principles that supported formal assurance, including unity of mechanism (doing a task all in one place) and separation of policy and mechanism. In addition, one of its design strengths was simplicity. For example, Multics' virtual memory design was useful for what it got rid of: buffers, inodes, puts, and gets, which were replaced by a unified name space. Also, backups were automatic and silent; Boebert used Multics from 1969 to 1988 and never lost a file.

Multics was based on a "large process model" of the movement of an execution point through code modules. This supports assurance arguments about system security. For example, in Multics, access rights were part of the segment descriptor word (used for the page table). This provides a "hardware lemma": You cannot avoid hitting the access bits, because they are integral to forming a hardware address.

Boebert also discussed online threats, using the term "Internet slime" to describe such problems as active content. What you want to have is a system such that the Internet slime cannot under any circumstances get access to the kernel. Following the Multics design, you set up an intermediate module between the slime and the kernel: slime can't call the kernel object directly, but has to go through a "gate" object. The intermediate module performs argument validation, after which it can provide access to the kernel object, which keeps the slime object safely away. In case a really bad slime object tried to fool the kernel object into calling back to the slime, the hardware ring would trap it and stop the disaster from happening.

Other good features of Multics included pointers, single copies of procedures, and heavy use of dynamic linking. A NASA study concluded that you should either change software a little every day or a lot once a year if you want to have a stable system. Because it supported online software updates, Multics was remarkably stable.

Boebert added that "we had some lousy ideas back then, but in our own defense, we never came up with anything as silly as security as a side effect of copyright enforcement." He concluded with a brief prognosis: "When an old fart comes up here, he's supposed to forecast the future optimistically, so we all go away with a nice feeling." However, he is not optimistic that we will ever see a system as rigorously engineered as Multics: The desire for "crap in a hurry" will prevent this in the foreseeable future, sadly.

One audience member asked how we could bring back the climate in which Multics was engineered. Boebert replied that he would try to reinstate the principles, rather than the system itself. He thinks that it would be nice if somebody would "do one [operating system] right for the sake of doing one right," as a learning experience.

ATTACK CONTAINMENT

Summarized by Stefan Kelm

A Virtual Honeypot Framework

Niels Provos, Google, Inc.

This year's first refereed paper was presented by Niels Provos, the author of honeyd, which he described during this talk. A honeypot can be defined as a computing resource that we explicitly want to have probed or attacked in order to study an attacker and his actions on our system. In terms of their implementation, honeypots can be divided into low-interaction vs. high-interaction honeypots as well as physical vs. virtual honeypots.

honeyd offers a framework for creating low-interaction virtual honeypots. It is able to simulate TCP, UDP, and ICMP packets and to adapt what Niels described as "different operating system personalities." Thus, honeyd simulates the TCP/IP stack behavior of a huge number of today's known OS implementations, thereby hiding the IP stack of the actual system honeyd is running on. This is realized by the so-called "personality engine," one of the core components of honeyd. The personality engine is based on nmap's fingerprint file: Anyone running nmap against a honeyd system will therefore get whatever OS has been configured to show up.

Another core component is the traffic dispatcher: Since honeyd does not itself intercept any network traffic, the traffic needs to be redirected to honeyd. The traffic dispatcher then has to decide how to handle any incoming packets. Since there are several ways to redirect traffic, honeyd is able to simulate not only a single IP address but complete network topologies, consisting of multiple hosts running different operating systems. Interestingly, honeyd seems to do very well performance-wise: On a standard PC, honeyd is able to simulate roughly 2000 TCP connections per second.

Niels went on to describe possible applications, namely, detecting and disabling worms as well as preventing spam. The author concluded that honeyd provides an efficient and scalable framework that deceives fingerprinting. Future work will be to enhance honeyd for further attack detection.

A number of interesting questions were asked by the attendees, two of which focused on honeyd in an IPv6 environment. Niels acknowledged that deploying IPv6 honeypots will likely be more expensive.

For more information, contact niels@google.com or see http://www.citi.umich.edu/u/provos/honeyd/.

Collapsar: A VM-Based Architecture for Network Attack Detention Center

Xuxian Jiang and Dongyan Xu, Purdue University

This talk can be seen as a follow-up to the previous one. Dongyan started by describing some of the problems of current honeypots, namely, lack of expertise, inconsistencies when operating multiple honeypots, and no central management. His solution is called Collapsar, a VM-based architecture which is founded on Lance Spitzner's honeyfarm idea.

Collapsar tries to integrate two goals: implementing a distributed honeypot presence within a network topology, while maintaining a centralized honeypot operation. It consists of three main functional components: (1) the redirector runs in each participating network and captures all traffic to unused IP addresses, which is then redirected to another component; (2) the front end acts as an interface between the redirectors and what is called the "Collapsar center"; (3) the virtual honeypots inside the Collapsar center are implemented as high-interaction honeypots and are therefore able to simulate different operating systems as well as popular services such as Sendmail and Apache. In addition, there are a number of different assurance modules, providing, for example, traffic logging and subsequent data correlation.

Dongyan next described the results of a Collapsar experiment they've been running. The honeypot architecture was deployed in a local environment that consisted of traffic redirected from five different networks with 40 honeypots running within the Collapsar center. During that experiment they were able to do a forensic analysis on incidents such as attacks on Apache or on XP. As to the performance, they measured TCP throughput as well as ICMP latency for both VMware and UML (user-mode Linux).

The authors observed that Collapsar even allows for more sophisticated analyses, such as stepping-stone identification and detection of network scans.

During the discussion one member of the audience remarked that Collapsar relies on the (bad) principle of "security by obscurity" in that the redirectors need to remain in the "dark IP space" (i.e., hidden from possible attackers) in order to function properly. This was confirmed by Dongyan.

For more information, contact dxu@cs.purdue.edu or see http://www.cs.purdue.edu/homes/jiangx/collapsar.

Very Fast Containment of Scanning Worms

Nicholas Weaver, International Computer Science Institute; Stuart Staniford, Nevis Networks; Vern Paxson, International Computer Science Institute and Lawrence Berkeley National Laboratory

Nicholas Weaver talked about a new and very fast method to scan for kinds of "Internet slime" in order to stop these worms efficiently. Static defenses have been and still are insufficient to stop worms from spreading in networks. The reaction needs to be automatic.

The algorithm Nicholas and his group have developed aims to implement worm containment, i.e., isolating a worm and subsequently stopping it from spreading further. Their work is based on the Threshold Random Walk (TRW), which has been modified for better hardware and software implementation possibilities. The basic idea is to use caches in order to keep track of incoming and outgoing connections; if a particular counter has been reached, connections start being blocked.

One interesting addition to the algorithm seems to be cooperation between said containment devices. If a device is able to use the status of other devices as another detector, the scanning results should be improved. The speaker concluded that not only are they able to enhance worm containment using their implementation, but they also gain a better understanding of particular worm attacks.

For more information, contact nweaver@icsi.berkeley.edu.

RFID: Security and Privacy for Five-Cent Computers

Ari Juels, RSA Labs

Summarized by Ashish Raniwala

Ari Juels, principal research scientist at RSA, gave an excellent presentation on security and privacy issues for RFID. RFID does not refer to any single device but to a spectrum of them, ranging from the basic RFID tag and EZ Pass to mobile phones. The talk focused on the basic RFID tag, which is a passive device that gets all its power from the RFID reader. In contrast to a bar code, RFID does not require line of sight between the tag and the reader. Additionally, RFID provides a unique ID for every object, as compared to bar codes, which only identify the type of object. An RFID has fairly limited memory (hundreds of bits), computational power (thousands of gates), and wireless range (few meters), making it hard to implement any real cryptographic functions or non-static keys.

RFID is already seeing many practical applications, starting from supply-chain visibility to anti-counterfeiting drugs. With push from such industry giants as Wal-Mart and Gillette and the decreasing cost of tags and readers (estimated to be 5 cents/tag and several hundred to several thousand dollars/reader by 2008), RFIDs are expected to become the physical extension of the Internet.

However, there are several security and privacy issues associated with RFID usage. An RFID tag can be clandestinely scanned to get personal information such as items carried on person. Limited computing resources make it hard to enforce strong access control on the RFID. Because of such concerns, RFIDs are already facing strong opposition, not just from customers but from corporations as well. For example, corporations need to worry about espionage and tag counterfeiting.

One approach to address the privacy issues is to "kill" the RFID tag as soon as the customer checks out items from the store. However, RFIDs are much more useful in their "live" state. Ari mentioned several novel applications that require that RFIDs stay alive post-checkout. For example, one can imagine a "smart" closet that can detect what clothes one has and suggest latest styles, or a "smart" medicine cabinet that can aid cognitively impaired patients, or automated sorting of recycled objects.

Ari suggests that it is hard to come up with useful solutions if one assumes the omnipresent oracle adversarial model. One needs to work with more realistic and weaker adversarial assumptions. Ari discussed three different approaches based on his own research. The first approach, termed "minimalist cryptography," is based on the observation that an adversary has to be physically close to the tag to be able to read it, and therefore can query it only a few times in any attack session. One can therefore store multiple pseudonyms on the tag, and return a different one each time the adversary queries. This makes it hard for the adversary to associate any tag with a particular object. The object-pseudonyms association can be known only to a trusted verifier. This approach can be further strengthened by throttling the query rate to prevent rapid, repeated scanning of the tag, and by refreshing the set of pseudonyms using a trusted reader.

The second approach is based on the idea of a "blocker tag," which can simulate all possible tags for any object. The blocker tag exploits the "tree-walking" anti-collision protocol used in ordinary tag-reader communications, and returns both "0" and "1" every time the reader queries for the next bit of the tag. This spams the reader with a large number of tags, protecting the privacy of the person. A blocker tag, however, needs to be selective in its blocking; for instance, it should only block purchased items carried by a shopper, not the unpurchased ones. Juels discussed use of a privacy bit that can be turned on or off based on whether the user wants the item to belong to the privacy zone.

The third approach uses a proxy RFID device, such as a mobile phone. The proxy device acquires the tags of the object and deactivates the original tag. All queries to the tag are then answered by the proxy device while enforcing the desired privacy policies. The proxy device can release the acquired tag and reactivate the original tag when it is about to leave the wireless range of the proxy.

RFIDs have led to a significant privacy debate, but at this point these security and privacy problems are more psychological than technological. Currently, RFID deployers can barely get the technology to work. For instance, UHF tags do not work well when close to the human body. It is hard to distinguish items in one shopper's cart from those in another's, because of the uncontrolled wireless range.

Ari concluded the talk by discussing some of the open issues for research: more realistic adversarial models for evaluating security techniques, and anti-cloning to prevent cloning of RFIDs. Ari pointed to *http://www.rfid-security.com* for more information.

PANEL: CAPTURE THE FLAG

Giovanni Vigna, University of California, Santa Barbara; Marc Dougherty, Northeastern University; Chris Eagle, Naval Postgraduate School; Riley Eller, CoCo Communications Corp.

Summarized by Rachel Greenstadt

The goal of this panel was to explore the utility of Capture the Flag competitions as pedagogical tools. These are competitions in which teams compete to defend their computers/data and attack those of the other team. A scoring bot periodically gauges the progress of the various teams and assigns points.

The panel, moderated by Tina Bird, got off to a rough start as Riley/Caesar—representing the Ghetto Hackers—was delayed. The Ghetto Hackers run the famous Capture the Flag competition at DefCon. He arrived by the end, but the panel began with the other panelists: Chris Eagle, associate chairman of the Naval Postgraduate School and a member of their Capture the Flag team; Marc Dougherty, who runs a Capture the Flag competition at Northeastern; and Giovanni Vigna, an associate professor of CS at UCSB who runs Capture the Flag competitions at his school and participates in the DefCon competition. It turned out Chris represented the winning team at this year's DefCon, and Giovanni, the second place team.

Marc Dougherty got his start in Capture the Flag competitions by participating in the competition at Northeastern and winning by exploiting a trivial weakness in the game setup. He decided to help fix the competition and has been involved in it ever since. He likes Capture the Flag competitions because they give students an opportunity to play with tools they wouldn't get to play with (legitimately) otherwise. At Northeastern, Capture the Flag is all-volunteer, not part of any class, and is all about fun. They run their competition attached to the Internet (as opposed to most other competitions, which have an air gap), but the students are careful. Marc's working on a modular system for running and scoring games so that they are easier to set up and run. He hopes to have it out, and available to the public, by October.

Chris has been involved with Capture the Flag for four years; this year his team won at DefCon. In describing their victory, he explained how the setup of the game can have unintended consequences. This year's DefCon was supposed to be all about offense—85% of the points were gained from offense, only 15% from defense—but Chris acknowledged that Giovanni's team had a better offense than his. However, the way the scoring bot worked was to access each system and then award points based on how many of your own and other teams' tokens you had. Chris's team won by waiting for Giovanni's team to hack a bunch of systems and then hacking theirs, stealing all their tokens and defending them well.

Giovanni concurred with this analysis and also mentioned that DefCon was tricky this year because the VM image of the system they were supposed to attack and protect was Windows, and who knows how to deal with that? Giovanni talked more about running a competition in a university setting. Capture the Flag competitions are a great tool to educate students in security. In this hands-on experience, the students learn a lot about buffer overflows, SQL injection, etc. You can also structure the competition so that you get useful data for your research, which is a nice side effect. The competitions help students develop teamwork and crisis management skills. All the panelists agreed that the most difficult part of running the competitions was developing a good scoring system. A competition requires a lot of time to set up initially, but running it becomes easier after that.

This was Caesar's fourth year involved in running the DefCon Capture the Flag competition. After the Ghetto Hackers won the competition the third time in a row, they decided to run it. This year they changed the system so that the scoring bot monitored who had the tokens of the various teams. Before this, scoring was just based on rooting a box, which is not an interesting measure per se, since it doesn't provide a realistic view of the way information is stolen today (at this point, Tina noted that two of the panelists' eyebrows moved). The competition was easier to prepare this year; they got people with a bunch of drinks coding nonstop for 10 days and it was done. All the panelists praised the way the DefCon qualifying round was run this year. The Ghetto Hackers are organizing a large, Internet-based contest, called Mega Root Fu. The Ghetto Hackers are willing to run competitions in exchange for airline and hotel costs and a weekend beer budget ($200–$300 of alcohol a day).

Fighting Computer Virus Attacks

Peter Szor, Symantec Corporation

Summarized by Wei Xu

Every month, critical vulnerabilities are reported in a wide variety of operating systems and applications. Computer virus attacks are quickly becoming the number one security problem and range from large-scale social engineering attacks to exploiting critical vulnerabilities. In this talk Szor discussed the state of the art in computer viruses and computer virus defense. He presented some promising host-based prevention techniques that can stop entire classes of fast-spreading worms, as well as worms using buffer overflow attacks. He also discussed in-depth worm and exploit analysis.

Szor first presented a short history of self-replicating structures. Modern computers started from von Neumann's architecture, in which programs are stored in primary memory and the CPU executes the instructions one by one. In 1948 von Neumann introduced Kinematon, a self-reproductive machine. Later Ed Fredkin used cellular automata to represent self-reproductive machines. In 1966, warrior programs, Darwin and Core Wars, were devised. In these programs, warriors fight each other, which is very similar to modern worms. John Horton Conway's 1970 Game of Life is unusual. The game follows a set of simple rules, but it can produce a remarkably large number of patterns. In the early seventies, self-replicating programs emerged. For instance, in 1971–1972, Creeper was born during the early development of ARPANET. In 1975, the self-replicating version of the Animal game was released, written by John Walker, founder of Autodesk, Inc. In 1982, Skrenta's Elk Cloner, a virus on Apple II, was developed. Peter showed a quick demo of Elk Cloner using an Apple II emulator.

Peter then presented several representative virus code evolution techniques. The complexity of virus threats has increased over time. In 1987, code encryption began to be used to hide the function of the virus code, and in 1990, code permutation was introduced into viruses. The number of permutations of a single virus has increased significantly. For example, in 2000 W32.Ghost had 10 factorial different permutations. Polymorphic viruses, which first appeared in 1991, can create an endless number of new decryptors that use different encryption methods to encrypt the constant part of the virus body. An even more powerful technique used by polymorphic viruses is external polymorphic engines (Polymorphic Engine Object, or MtE). Emulation provides a very good approach to detection of this type of virus.

Metamorphic viruses are more difficult to detect. Here "metamorphic" means "body polymorphic." Unlike other polymorphic viruses, metamorphic viruses do not require a decryptor. Instead, the virus body is "recompiled" in new generations, and the virus code can shrink and expand itself. Source code mutation (encrypted source code compiled into slightly different but equivalent encrypted source code) is usually used in metamorphic viruses (e.g., W32.Apparition). Equivalent instruction sequences are one s uch technique. There are metamorphic viruses for Windows operating systems, including the recent Windows .NET framework (MSIL.Gastropod in 2004). Surprisingly, there are also multi-platform metamorphic viruses, such as {W32, Linux}/Simile.D, which infects both Windows and Linux systems.

There is a class of "undetectable" computer viruses. W95.Zmist is one example. This virus is an entry-point-obscuring virus that is metamorphic. Moreover, the virus randomly uses an additional polymorphic decryptor. Zmist also supports a unique new technique: code integration. The Mistfall engine contained in the virus is capable of decompiling portable executable files to their smallest elements. Zmist first moves code blocks out of the way, inserts itself into the code, regenerates code and data references, including relocation information, and rebuilds the executable.

Next Szor talked about modern worm attacks. Nowadays, we have another important problem: fast-spreading worms. The frequency of worm attacks is increasing, and worms are getting faster than update and patch deployment. Meanwhile, known vulnerabilities are on the rise. There are many incidents of worms that caused major DoS attacks; for instance, the Blaster worm contributed to a major blackout. Here are several famous worms: Blaster (August 2003), Code Red (July 2001), Nimda (September 2001), Slammer (January 2003).

The Code Red worm is based on a buffer overflow attack on Microsoft Internet Information Server (IIS). The worm uses GET requests to trigger the buffer overflow vulnerabilities in IIS.

The Slammer worm is another buffer overflow–based attack targeting Microsoft SQL Server. The Slammer code is amazingly small, only 376 bytes.

To detect these types of worms, we can use generic IDS signatures characterized by the sizes of buffer and input strings as well as other important attack patterns. Below is a rule to detect Slammer:

```
alert udp any any -> any
1434 (msg: "MS02-039
exploitation"; content:
"|04|"; offset 0; depth:1;
dsize>60)
```

where 60 is calculated by size_of_buffer - (string_size1 - string_size3 - terminating_zero) = 128 - 40 - 27 - 1 = 60.

Szor gave the details of how Blaster works. Basically, Blaster exploits the target on port 135/tcp of Windows XP/2000 and uses the TFTP protocol to upload the worm code onto the victim systems.

Szor then presented various worm-blocking techniques.

In a host-based layered security model, we have layers such as personal firewall, anti-virus protection, host-based IDS, and OS/application layer prevention or access control. The network can also be a part of this model. Outbound SMTP worm blocking is an example of network protection. In this case, emails sent by an SMTP client are first redirected to a scan manager for virus scanning before being passed to the SMTP server. The scan manager can use a NAV client and consists of a decomposer and anti-virus engines.

Digital Immune System can provide worm/virus protections for a large-scale network. In this system, the customer-side AV client communicates with a quarantine server, which itself talks to the vendor-side servers through a customer gateway. The vendor-side servers include automated/manual analysis centers and definition servers. Anti-virus researchers interact with these servers to identify new viruses/worms and provide detection mechanisms.

There are many techniques for blocking buffer overflow attacks: code reviews, security updates (patches), compiler-level extensions (e.g., StackGuard; ProPolice; and Buffer Security Check in MSVC .Net 7.0/7.1), user-mode extensions (e.g., Libsafe), kernel extensions against user-mode overflows (e.g., PAX, SecureStack), and OS built-in protections (e.g., Solaris on SPARC).

An NX (non-executable) page attribute on AMD64, EM64T, and updated Pentium 4 can be used to prevent buffer overflow attacks. When an NX-marked page is executed, an exception is generated. The Data Execution Prevention (DEP) in Windows XP SP2 takes advantage of this feature. Note that there are already viruses that are aware of DEP.

In order to block worms, we can also deploy other techniques such as exception-handling validation, injected code detection (shell-code blocking), and self-sending code blocking (send blocking). These techniques can stop attacks after a buffer overflow or prevent viruses/worms from propagating.

Szor showed demos on blocking shell code in both Windows and UNIX systems. He also showed demos on blocking various worms. It is worth mentioning that the Witty worm made use of exploitations of third-party software (Internet Security Systems' security products).

In the end, Szor pointed out some future threats from viruses/worms.

Worms can do secure updating. For example, the Hybris worm has more than 20 plug-ins, and these plug-ins can be downloaded via secure channels by the worm itself.

Worms can communicate. They can export/import user accounts/passwords, payloads, mutation engines, and exploit modules.

Viruses for mobile phones, smart phones, and pocket PCs will become a huge problem. In fact, there already exist proof-of-concept viruses for these platforms.

Q. Have you seen any viruses on mobile devices that spread by means other than Bluetooth?

A. Haven't seen such viruses. But we expect to see viruses that leave a backdoor on mobile devices and use SMTP to send out emails with attachments or invitation SMS messages to infect other devices.

Q. How would I get access to proof-of-concept viruses?

A. No public accesses. We get them by submissions.

Q. What is the meaning of "proof-of-concept" viruses?

A. These are viruses that do not want to hide themselves.

Q. How can one detect Zmist today?

A. It is hard. But we can detect the virus using a disassembler.

Q. Is there convergence between anti-spam and anti-virus efforts?

A. We have recognized that anti-spam techniques are very important. We are investigating this.

PROTECTING SOFTWARE

Summarized by Chad Mano

TIED, LibsafePlus: Tools for Runtime Buffer Overflow Protection

Kumar Avijit, Prateek Gupta, and Deepak Gupta, IIT Kanpur

Kumar Avijit presented a two-pronged approach for protecting both new and legacy code from the "menace" called buffer overflow. This is accomplished by using TIED (Type Information Extractor and Depositor) and LibsafePlus.

TIED uses debugging data generated by the compiler to create a data structure containing the size information of all global and automatic buffers. This data structure is inserted into the program binary to be used at runtime. LibsafePlus is an extension of the Libsafe tool which provides more extensive bounds-checking features. LibsafePlus identified all 20 buffer overflow attacks in the Wilander and Kamkar test suite, while Libsafe identified only six.

The overhead associated with these tools is around 10 percent. The code is available at http://www.security.iitk.ac.in/projects/Tied-Libsafeplus.

Privtrans: Automatically Partitioning Programs for Privilege Separation

David Brumley and Dawn Song, Carnegie Mellon University

David Brumley explained that attackers target privileged programs because "it gives them a bigger bang for their buck." Programs can be partitioned into two separate entities to increase protection. This prevents a bug in the unprivileged part of the program from giving access to a privileged area. The privileged program, or monitor, handles sensitive processes, such as opening /etc/passwd. The unprivileged program, or slave, handles all other processes. The important result of partitioning is a small monitor, which means the trusted base is easier to secure.

Partitioning of programs is traditionally done manually. Mr. Brumley presented the first approach to automatic privilege separation. This is accomplished using a tool called Privtrans. Variables and functions in the source code must first be annotated to identify the privileged parts of the program. For the programs tested this took between 20 minutes and two hours. Privtrans takes the annotated source and creates separate monitor and slave source code. Communication between the two programs is implemented with RPC.

Privtrans is able to obtain results similar to manually partitioned code. The associated overhead is reasonable for the increase in software security.

Avfs: An On-Access Anti-Virus File System

Yevgeniy Miretskiy, Abhijith Das, Charles P. Wright, and Erez Zadok, Stony Brook University

Yevgeniy Miretskiy presented Avfs, which is a true on-access virus-scanning file system. In contrast to on-open, on-close, and on-exec scanning, which run in user space, Avfs runs in kernel space and provides incremental file scanning, which prevents infected files from being written to disk.

Avfs is based on the ClamAV virus scanner, which was enhanced and renamed Oyster. The enhancements improve the scalability from a linear function to a logarithmic function. The important features of Avfs are that it detects viruses early to prevent file system corruption, it avoids repetitive unnecessary scanning by implementing partial scanning, it works transparently in kernel space, and it can be ported to many file systems.

I Voted? How the Law Increasingly Restricts Independent Security Research

Cindy Cohn, EFF

Summarized by Rachel Greenstadt

Cindy Cohn used the work Avi Rubin and his colleagues did examining the Diebold voting machines to illustrate a problem facing security researchers today: In order to do our research, we need "her or someone like her" to answer our legal questions.

The first hurdle she often faces when dealing with these matters is people questioning the need for independent security research. She often has to explain that the vendor might not otherwise be motivated to fix security problems and that, since we are all connected, good security is in the public interest. The insecure system in question in this case was the Diebold voting software. The source code was leaked on the Internet, and researchers at Johns Hopkins and Rice did an independent security audit. They found serious problems.

The talk focused on four areas of the law where the Diebold study could be called into question. These were copyright law, the DMCA, the Computer Fraud and Abuse Act, and trade-secret law. In the case of copyright law, Diebold owned the copyright to the code. In order to study the code, the researchers needed to make copies, facing as much as $150,000 in fines for infringement. In this case, the researchers had a good case for making fair use of the code, since it was for critical and scientific work and didn't impede the market. However, the law is structured poorly for protecting fair use, as there is no cut-and-dried way to tell whether an action qualifies as fair use, so you need to hire a lawyer.

Next came the DMCA. The DMCA prohibits people from circumventing a technological measure that controls access to a copyrighted work. The scientific exception is effectively useless. Some of the files were encrypted zip archives, so the EFF instructed the researchers only to audit code that had been unzipped by others. Members of the audience expressed incredulity that a zipped archive could be considered an effective control mechanism. Another issue raised by the DMCA is publishing findings. It is unclear how far this goes: DeCSS? source code showing how to bypass some DRM? a high-level description of such code? an academic paper? This last case was at issue in the SDMI challenge paper presented at USENIX '01 Annual Technical Conference. The RIAA made threats but then backed down, so no legal precedent has been set.

The Computer Fraud and Abuse Act is the generic federal computer intrusion law. It states that unauthorized access to a protected computer is illegal. Even though the code was publicly available on Diebold's servers, someone probably broke this law to access it. There was a concern that Diebold might claim an intruder stole the code and handed it to the researchers. That would be difficult since the code was so widely available on the Internet. But the researchers had to depend on code falling out of the sky.

The last issue that often comes up is trade-secret law, which basically states that a company's desire to keep any particular information secret has to be respected and that anyone who publishes that information is liable. Many vendors would love to claim that the security flaws in their products are trade secrets. Fortunately, reverse engineering is allowable. However, not all security flaws can be found through reverse engineering, and EULA licenses often prevent reverse engineering. This institutionalizes security through obscurity.

The researchers in the Diebold situation finally got enough clearance to do their research, but there were a lot of legal pitfalls, and EFF needs support. Cindy urged the audience to support HR 107, which provides a fix to the DMCA, and said that it is important to talk about the need for security research with people who are nontechnical. A good way to frame the issue is in terms of consumer protection: people need to be protected from companies that will sell you snake oil security and buggy software. Avi Rubin mentioned that everything turned out okay in their case, but would she advise other researchers to take the same risk? Cindy responded that, at the end of the day, she does not tell people what to do. It's her job to lay out the risks, costs, and benefits and whether EFF will represent them. Great advances in the law have happened because people were willing to take chances and bear it if they lost. The history of the law demonstrates that people are rarely given rights, that they must take them. She can tell you what the law says now, but then you have to make your decision.

PROTECTING SOFTWARE II

Summarized by Stefan Kelm

Side Effects Are Not Sufficient to Authenticate Software

Umesh Shankar, Monica Chew, and J.D. Tygar, University of California, Berkeley

This talk was somewhat different from most of the other ones, in that it didn't present a solution to a security problem but presented an attack to a security solution that had been presented at the previous security symposium.

Umesh presented a number of problems with Genuinity, a system allegedly able to authenticate remote systems without using trusted hardware. Since the Genuinity source code was not available to the authors, they based their attack scenarios on what had been published already. The basic idea behind Genuinity is the computation of checksums by remote systems, which subsequently gets verified by the authenticating system. The authors described and implemented what they called a substitution attack: They computed the correct checksum quickly while running non-approved (arbitrary) kernel code. Thereafter, the original data in the computation needed to be substituted.

Umesh even went to so far as to argue that remotely authenticating systems will not be possible at all without some form of trusted hardware. Moreover, he questioned the availability of useful applications except, e.g., set-top game boxes.

For more information, contact ushankar@cs.berkeley.edu or see http://www.cs.berkeley.edu/~ushankar/research.

On Gray-Box Program Tracking for Anomaly Detection

Debin Gao, Michael K. Reiter, and Dawn Song, Carnegie Mellon University

In this more theoretical talk, Debin Gao presented work on anomaly detection systems. Today there are a number of systems trying to detect anomalies by closely monitoring system calls; these systems usually can be grouped into white box, black box, and gray box systems. The motivation behind this work was to perform a systematic study on existing systems.

During the analysis Debin and his team discovered that all the current systems can be organized along three axes: (1) the information as extracted from the process on each system call; (2) the granularity of atomic units; and (3) the size of the so-called "sliding window." In doing so they try to focus on all of the 18 resulting areas instead of looking at only one, pointing out advantages and disadvantages of one area over another. Detailed results are described in the proceedings.

In the Q&A session two attendees remarked that this work does not take system-call parameters into account, which the author confirmed. For more information, contact dgao@ece.cmu.edu.

Finding User/Kernel Pointer Bugs with Type Inference

Rob Johnson, David Wagner, University of California, Berkeley

The final presentation of this session focused on the issue of pointers that are passed from user applications into kernel space and that might cause security problems. Misusing this "feature," an attacker might be able to read or write to kernel memory, or simply crash the machine. According to Rob, this has been and still is a persistent problem in the Linux kernel (as well as other kernels).

The solution presented by the authors uses type qualifiers that enhance the standard C language types by defining both a kernel and a user type qualifier. This has been implemented using CQUAL, a type-qualifier inference tool.

In an experimental setup, they tested their enhancements against several versions of the Linux kernel, using CQUAL in verification as well as bug-finding mode. Of the many interesting results, they presented 17 different security vulnerabilities, some of which had escaped detection both by other auditing tools and by manual audits. Moreover, all but one of those bugs was confirmed to be exploitable by an attacker. As an additional result, not in their paper, they reported that their analysis of Linux 2.6.7-rc3 revealed that although some bugs had been fixed, they were able to find seven new security vulnerabilities in only two days.

Someone asked why programmers are not using the kind of solution presented here even though similar work has been available for quite some time. Rob replied that he thinks this is beginning to change.

For more information, see http://www.cs.berkeley.edu/~rtjohnso/.

Metrics, Economics, and Shared Risk at the National Scale

Dan Geer, Verdasys, Inc.

Summarized by Serge Egelman

Dan Geer's talk centered on risk assessment as applied to national security. He covered the current state of risk assessment, the nature of risk, how various models are used to measure risk, and predictions for risk in the near future.

Geer first explained how risk assessment is related to engineering; getting the problem statement correct is extremely critical. The correct problem cannot be solved unless the right questions are asked and the problem is thoroughly understood. In terms of national security, the questions are centered around what can attack our infrastructure, what can be done about it, what the time frame is, and who will care about it.

Because of the Internet and other enabling technologies, our society has greatly increased the amount of information that is available. While this increases productivity, it also increases risk since, in an interdependent society like ours, it is now increasingly easy to stumble into different locales. The technologies are still following Moore's Law, yet the skills needed to properly use this new power are unable to keep pace. This divergence is creating a dangerous situation with broad implications.

Geer mentioned sitting in on a risk management discussion at a major bank. The key to managing their risk is knowing exactly how much risk to take. Not taking enough risk does not take full advantage of potential returns (thus losing money), whereas taking too much risk can result in insolvency. The biggest problem for them is finding this median. In finding the right amount of risk, scenarios are discussed. These scenarios cover every conceivable factor, as risk is the result of many interdependencies. This system is the basis for our banks and financial markets, and it largely works, for one reason: There is zero ambiguity about who is responsible for taking the risks. And this is a problem for national security because it is not very clear who owns the risk.

Along this line of reasoning, online risk is difficult to calculate because the Internet is a new and unique domain. The economic models function because the institutions are fairly static and predictable, whereas the Internet crosses many boundaries and jurisdictions. John Perry Barlow suggests that the Internet is separate from other jurisdictions, while others argue that it belongs to every jurisdiction. Attacks on the Internet are easy because the HTTP transport model allows the attacker to concentrate on the attack itself, as he or she does not need to worry about propagating it. Software companies are increasingly relying on HTTP for communication with applications, thus allowing attacks to be more concentrated. Additionally, this will increase the total amount of traffic on HTTP, which will make content filtering infeasible.

Economics have dictated a shift toward data. Corporate IT spending on data storage has been increasing every year, from 4% in 1999 to 17% in 2003. The total volume of data has been doubling every 30 months. Because of this, the focus of security will shift toward protecting the data. Viruses, spam, and online theft (whether it be identity or economic) have greatly increased. Users respond with complaints, and legislators respond with an increase in laws governing online security and privacy. But much of the response is ill-conceived; HIPAA attempted to account for technology change, but resulted in a law that was totally unreadable.

On the national scale, there are many considerations for mitigating risk. We create unique assets to decrease ambiguity; for example, there is only one DNS system, since having multiple ones will defeat the purpose. But this creates a huge risk in that it offers a single target. The best counter to this is replication, creating multiple homogeneous nodes that all work together. On the other hand, replication can result in cascade failure; nodes can be used to propagate attacks. Creating this monoculture has a direct effect on exploits, since only one is needed to take out entire networks of machines. Additionally, the amateur attackers create smokescreens for the professional attackers.

Currently, the best solution we have for online attacks is field repairs. Software is patched when a vulnerability is found. This has many implications, the first of which is liability. Software is sold "as is," and thus the user agrees not to hold the manufacturer liable for any problems if the product updates are not applied. Automatic updating of software has been mandated in many license agreements for this same reason. At the same time, applying patches is not without risk. As Fred Brooks described, fixing flaws in old software invariably introduces new ones.

Geer predicts that within the next 10 years, advances in traffic analysis will greatly increase, just as advances in cryptography have increased over the last 10 years. Additionally, increasing threats will result in decreasing security perimeters. Currently firewalls are used to secure a corporate network; this will shift toward mechanisms to protect individual data. Security and privacy are currently being treated similarly; this will also change in the future.

Geer discussed further challenges in the next 10 years. Among those were eliminating large-scale epidemics and minimizing the amount of skill needed to be safe. He proposes creating commercially available tools for creating certifiable systems. Finally, he believes that information security needs to be as good as or better than financial security. But to accomplish this, better metrics are needed to answer specific assessment questions.

Because of the immediacy of this need, there is no time to create a system from scratch. We must steal models from other fields such as public health, insurance, financial management, and even physics. These models must take into account information sharing and must place values on the information. This amounts to calculating both replacement and future values. Currently the basis for these are the black market values: email address, chat screen names, and pirated software are all sold. In lieu of measuring security, we can simply assign risk in much the same way that credit card companies do (i.e., $50 liability on fraudulent purchases). These metrics are not free, however. There is always the tendency for government or other entities to impose strict controls.

In conclusion, Geer stressed that, rather than using words to describe problems, we need to do quantitative analysis. Unknown vulnerabilities are secret weapons of the attackers. The absence of a major incident says nothing about a decrease in risk. All of this is further ensuring that tradeoffs will be made between safety and freedom.

THE HUMAN INTERFACE

Summarized by Nick Smith

Graphical Dictionaries and the Memorable Space of Graphical Passwords

Julie Thorpe and Paul van Oorschot, Carleton University

Graphical passwords can be a more secure authentication mechanism than traditional text passwords, because they offer a wider set and could possibly effect a positive change on the percentage of the password set that a typical user will choose. It may also be easier for users to memorize an image than a text password.

The idea proposed is to have a user draw an image on a grid (larger or smaller to increase or decrease size of the password set) and to track the start and end position of each pen stroke to create an image definition. In this way, identical images could possibly be drawn in several different ways, which further expands the password set. The idea does suffer from the same problem as text passwords, though, which is that users are more likely to draw certain patterns (e.g., symmetric images along center axes, common images, low number of pen strokes, similar drawing sequences).

Regardless of drawbacks, which are unavoidable in any solution, graphical passwords are shown by Thorpe and Oorschot to be potentially more useful and secure than text passwords.

On User Choice in Graphical Password Schemes

Darren Davis and Fabian Monrose, Johns Hopkins University; Michael K. Reiter, Carnegie Mellon University

Recognition of images, specifically, of human faces, can be used to replace text passwords for authentication. Humans are especially good at pinpointing minute and detailed characteristics in the human face, especially in those of a person's native culture.

There are two proposed schemes: the "Face" scheme, in which a user chooses a variable number of faces from a single provided set; the "Story" scheme, where a user chooses one picture (not all of which are faces) from each of several provided sets (recorded in sequence). In a small, 154-student study done within the university, it was found that the "Story" scheme proved much more secure (i.e., more difficult to guess) than the "Face" scheme.

One drawback to these schemes is that certain races/cultures/genders tend to choose the same type of pictures in a set (supermodels, cute animals, etc.). Also, although the "Story" scheme proved superior to the "Face" scheme, some students found it hard to remember the sequence of pictures even though they could identify all of the correct images.

Design of the EROS Trusted Window System

Jonathan S. Shapiro, John Vanderburgh, and Eric Northup, Johns Hopkins University; David Chizmadia, Promia, Inc.

The goal of this system is to impose MLS information flow controls into the X Window System. The idea is that each window's information should not be available to all other windows, but only to trusted windows (such as those within the same process). For example, common features such as cut&paste and drag&drop currently use bidirectional communication between windows, which violates the security measures of the system. Workarounds were developed to make the communication unidirectional. Also, the storage buffers used for each window must now be managed by the client rather than the window server (in this case, X). Future work anticipates the porting of current GUI toolkits.

Exploiting Software

Gary McGraw, Cigital

Summarized by Chad Mano

Gary McGraw presented an eye-opening view into the world of software security. Traditionally, security is thought of as an infrastructure problem, not a software problem. Also, many believe that cryptography is the key to providing security. However, as McGraw puts it, there is no such thing as "magic crypto dust."

The security issue that needs to be addressed concerns the software, not the infrastructure. McGraw described the "Trinity of Trouble": connectivity, complexity, and extensibility, which are the underlying reasons why developing secure software is a difficult task.

- Connectivity: The Internet is everywhere, and most software is on it.
- Complexity: As the size of the code increases, so do the number of vulnerabilities.
- Extensibility: Systems evolve in unexpected ways.

Two things that make software vulnerable to attacks: bugs and flaws. Bugs are the simple, easy-to-find mistakes in the code. Flaws are more complex problems that can be the result of mistakes in code and design. We traditionally work on fixing bugs, whereas attackers don't care whether it's a bug or a flaw that allows them to exploit software.

Educating new software developers is one of the keys to improving security. Traditional software engineering classes are boring, but that does not have to be the case. Students should be taught how to break software the way hackers do so they will better understand how to build secure software. Breaking stuff is fun and makes for a more invigorating class, as well as providing a valuable learning experience.

Functional testing and security testing are not the same thing. Functional testing looks at a program as a black box and tries to determine whether the output is correct. Security testing needs to get inside the code. Hackers use tools such as decompilers and disassemblers to understand the control flow of a program. Security testing cannot be used to build security into a program. Security must be built into the design of the program, so that security is truly inside the code.

See McGraw's book, co-authored with Greg Hoglund, *Exploiting Software: How to Break Code*, for more information on this topic.

PANEL: PATCH MANAGEMENT

Crispin Cowan, Immunix; Marcus Ranum, Bellwether Farm; Eric Schultz; Abe Singer, SDSC; Tina Bird, InfoExpress

Summarized by Eric Cronin

Crispin Cowan began the presentations by looking at the tradeoff between safety and stability inherent in patching. Patching too late leads to getting hacked; patching too soon can lead to problems from buggy patches. For a given vulnerability these two trends move in opposite directions as time passes (bugs decrease, compromises increase), so there should be some optimal point, where these two curves cross, at which to apply the patch. Unfortunately, from studying historical information, Cowan found that on average it takes 10 days after being released for the final stable revision of a patch to arrive—up to 30 days in some cases. Attacks, on the other hand, are beginning within hours of a vulnerability being published. In reality, the two curves never intersect, so from a security standpoint patch management is a doomed cause. Cowan instead suggested that patching be looked at as a maintenance activity and alternative strategies be used for security. The first alternative proposed was to only run perfect software, which has some problems in practice. The second alternative was to rely on intrusion detection and intrusion prevention technologies, such as the fine products sold by Immunix.

Tina Bird related some of her experiences as system administrator at Stanford during the MS Blaster attack. On July 16 the vulnerability exploited by Blaster was announced. On July 18, proof-of-concept attack code became available. A month later, on August 18, Blaster hit the Internet. Despite this month-long window during which users could have patched, 8,100 Windows machines at Stanford were compromised. Many users had not patched because they did not want to break working systems. The CVE (Common Vulnerabilities and Exposures) reports announcing patches make it very difficult for users to gauge the criticality of a patch. From her experience, Bird thought the best solution in an environment like Stanford's was to remove free will from the users and use automated update systems to force patching. Bird also noted that by their nature, the most dangerous vulnerabilities are those against network services that are remotely accessible, that are on by default, and that do not require authentication. These same properties also make it very easy for administrators to scan their networks periodically and locate unpatched hosts.

Eric Schultz provided some insight on the view of patch management from "the other side." While at Microsoft he had a part in every service pack and security patch released. One strategy used by Microsoft was to remove all references from service pack release notes for fixes to previously undocumented/unannounced security fixes, in order to lessen the risks to those not applying the service pack. Microsoft usually saw a flurry of activity immediately after a patch was announced, and then little activity until right after a worm hit, at which time there was a spike in purchases of patch management solutions.

Abe Singer briefly talked about a major security attack suffered by SDSC last spring. Of note, none of the machines involved in the attack was running Windows; the vulnerabilities were all against *NIX hosts. A contributing factor to the vulnerability was that Sun's patch system indicated that kernel patches had been applied but the machines had not been rebooted to use the new kernel. Without accurate reporting it is very difficult to determine what the vulnerability status of hosts actually is.

The final presenter was Marcus Ranum, who concurred with Cowan's observation that patch management was the wrong strategy for security. He went one step further, however, and insisted that patching was a fundamentally stupid thing to do. Production Systems 101, he stated, is "If it's working, don't f— with it." He never patches his own systems, and he never changes his passwords (several of his passwords were then provided to the audience). Instead, he solved his security problem by configuring his network so that he doesn't have to care about new vulnerabilities. Corporations want the conflicting features of cheap and good software as well as unsegregated Internet-facing hosts and security. His solution is to run software that is less questionable than the rest of the herd and to run even that software within sandboxes and with extra protection. He concluded by stating that securing systems is engineering, while patching systems is anti-engineering.

In the Q&A, Steve Bellovin asked Ranum whether it was reasonable to expect every user to write her own "better than the herd" Web browser, for example. The panel answered that when you need to run commodity software such as a browser, you shouldn't run it on a machine that you care anything about; the techniques the panel had presented were suitable for servers. Another audience member pointed out that by properly isolating servers with chroot and systrace it is possible to run after a vulnerability without patching and still be secure. Another point raised was that most machines on a network are desktops and not servers, and that these machines are rarely professionally managed. When these machines are attacked, the collateral damage can impact the network and even the most hardened server. The panel agreed with this observation, and also noted that it is often wise to treat these unmanaged desktops (and laptops) as being just as dangerous as remote hosts on the Internet, and placing them outside the innermost firewall surrounding critical servers.

Military Strategy in Cyberspace

Stuart Staniford, Nevis Networks

Summarized by Marc Dougherty

Stuart Staniford presented his predictions for a hypothetical cyberwarfare scenario. There has not yet been a real cyberwar, but Staniford believes the tactics and technologies necessary will develop rapidly in the wake of real cyberwar. Staniford describes a scenario in which China invades Taiwan. The US dispatches a few carrier groups to assist the Taiwanese, causing substantial damage to the Chinese invasion force. In order to eliminate US involvement, China opens the North American theater with a cyberattack.

Staniford theorizes that in order to have sufficient results, the Chinese must disable two critical infrastructures. Staniford believes the number of targets necessary to disable an infrastructure is on the order of one hundred. The attacker not only must compromise the security of internal corporate networks, but must then gain access to the SCADA control systems in order to disable the target. Staniford estimates that a single attacking battalion could range from 150 to 1000 attackers.

Only a nation-state could muster the extensive resources necessary to create such a tremendous force. The attacking force is likely to be extremely disciplined and well trained. Because of this, Staniford believes that defensive forces must be organized and trained like a military regime, especially in the case of critical infrastructures.

SECURITY ENGINEERING

Summarized by Marc Dougherty

Copilot — A Coprocessor-Based Kernel Runtime Integrity Monitor

Nick L. Petroni, Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh, University of Maryland

Nick Petroni described a Kernel Integrity Monitor known as Copilot, which resides on a PCI card. The goals of Copilot are to create a system that is compatible with commodity hardware, effective in detection as well as performance, and isolated from the host, and that cannot be tampered with in the event of a compromise. The current system functions on commodity x86 Linux systems without modification.

Copilot monitors a number of data points, including the list of loaded kernel modules and a hash of the Linux kernel text. The results of these checks are compared to a known good state, and in the event of a change, it is reported to the administrative system, to which the Copilot PCI card connects directly.

In an experimental test of Copilot's capabilities, it was able to detect all 13 of the rootkits attempted. The performance penalties of using Copilot are also relatively minor. When checking every 30 seconds, there is only a 1% performance penalty. Continuous checking, however, showed a 14% penalty, which is believed to be caused by bus contention.

Further work for the Copilot team includes predicting memory footprint based on a binary, and instituting further checks on kernel data such as the process table and open file descriptors. Copilot can also currently be extended to support dynamic data.

Fixing Races for Fun and Profit: How to Use access(2)

Drew Dean, SRI International; Alan J. Hu, University of British Columbia

Drew Dean presented a probabilistic solution to the age-old problem of race conditions. Dean's solution deals particularly with the problem of setuid programs using the access rights of the real user. The `access()` system call was created for this purpose, but due to the non-atomic nature of the call, it is still vulnerable to a TOCTOU attack (time-of-check to time-of-use).

Dean's approach involves applying "hardness amplification," a technique commonly used in the cryptography realm, to make winning this type of race more difficult. The system described uses a coefficient K to determine the strength of the system. When opening a file, `access()` is called, and the file is opened. Instead of using that file descriptor, the following steps are performed K times.

- Random delay
- Call `access()`
- Random delay
- Re-open the file
- Verify that the device, inode, and `fstat()` results match
- Close the re-opened file

This process relies on `access()` producing no side effects, `open()` being idempotent, and `fstat()` not being vulnerable to a race condition. The result of this procedure is that an attacker must win 2K + 1 races in order to achieve his goal. Dean recommends a K of 7 for single-processor machines, and a K of 10 for a reasonable level of security.

Network-in-a-Box: How to Set Up a Secure Wireless Network in Under a Minute

Dirk Balfanz, Glenn Durfee, Rebecca E. Grinter, Diana K. Smetters, and Paul Stewart, PARC

Diana Smetters presented a PKI-based wireless security solution modeled on the 802.1x EAP. The barrier preventing widespread use of EAP is the configuration, which involves typing large hexadecimal strings into the clients and the APs.

The goal of NiaB is to create a smarter access point that handles the generation of certificates and exchange this new authentication information through a second communication channel. By using a short-range method such as IR for this second channel, wireless security becomes a matter of physical security.

The NiaB architecture is divided into four parts:

- Wireless Access Point
- Enrollment Station
- Certificate Authority
- Authentication Server

When joining a network, the user interacts with the Enrollment Station portion of the system, which talks to the CA to autocreate client keys. Once the keys have been exchanged, the user can safely communicate with the Authentication Server element over TLS, since the public key digest is also exchanged via the secondary communication channel. This communication is done using a custom protocol called EAP-PTLS.

Smetters also noted that for enterprise use, the architecture can be separated to allow a centralized Enrollment Station, CA, and Authentication Server. Experimental results show that users can connect to a NiaB network in under one minute, demonstrating that wireless networking can be usable as well as secure.

Design and Implementation of a TCG-Based Integrity Measurement Architecture

Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn, IBM T.J. Watson Research Center

Reiner Sailer presented an integrity detection system based on the work of the Trusted Computing Group. The TCG system relies on a special chip that stores information about the boot process. The system presented by Sailer uses this same TPM chip to detect tampering with important system files. TPM chips are already installed in most IBM laptops.

This system uses the TPM chip to store SHA1 checksums of important system files. These checksums are then digitally signed by the chip. The checksums are also available to the kernel of the host OS.

The software portion of the current implementation consists of a kernel module which performs the above checksumming before opening each file. Experiential results show that a clean hit takes only .1 millisecond, and the opening of a new file takes only 5 milliseconds plus the time required to perform the SHA1 checksum.

This work is included in the TCG Linux project. More information is available at http://www.research.ibm.com/secure_systems_department/projects/tcglinux/.

What Biology Can (and Can't) Teach Us About Security

David Evans, University of Virginia

Summarized by Alvin AuYoung

Note: Slides from the talk are available at http://www.cs.virginia.edu/~evans/usenix04.

Nature provides a fascinating example of how to design computer systems. David Evans presented analogies between nature and computer systems, highlighting both the potential and the limitations of using nature to guide the design of secure computer systems.

Approximately 99.9% of species in nature become extinct. Evans emphasized the slim odds of survival faced by nature's species, who must adapt in order to survive their predators and other attacks in nature. He drew a parallel between attacks faced by these species and well-known attacks faced by today's computer systems. For example, a lion chasing down a gazelle might be similar to a "brute force attack," where an attack is conducted by sheer force. A Bolas spider attracting male moths by emitting chemicals that mimic pheromones of female moths is similar to a communication integrity attack.

Species that are able to survive against predators in nature and various ecological phenomena are examples of secure and robust biological systems. Understanding how these species have evolved can help us design secure and robust computer systems.

Biological systems are too complex to understand analytically. Instead, Evans suggests that we should learn from the qualities of existing species in nature that have survived over time. There are four common traits observed in such systems: expressiveness, redundancy, awareness of surroundings, and localized organization. If we were to mimic these characteristics, could we also build programs that survive both its adversaries and the test of time?

Amorphous computing, swarm programming, and autonomic computing are examples of research that attempts to use these characteristics to design computer systems. In particular, Evans discussed the amorphous computing project from the lab of Radhika Nagpal, where the goal is to design a system in which a potentially large number of simple and possibly unreliable pieces can cooperate to create a system with complex and well defined behavior. An example of this is Nagpal's work with origami mathematics. In this work, a set of identically programmed cells forms a sheet of (virtual) paper, and the goal is to have these cells cooperate using only local interactions to form a complex structure such as an origami shape (e.g., half of the cells will "move" to one side to simulate a fold in the paper). Although this is an example of how to design a robust system that captures some of the qualities of nature's systems, there is still a long way to go before we understand how to use these qualities to design secure systems for real-world applications.

Another place where the principles of nature's systems can be applied to computer systems is computer immunology. Identifying intrusion and removing viruses in computer systems is similar to how our immune system wards off attacks. In nature, diversity of receptors within species allows them to survive strains of specific pathogens. As a result, a large number of the members of a species can survive. In computer systems there has been a push toward providing system diversity. A commonly cited example is in operating systems, where over 90% of the machines in the world use Microsoft Windows. Largely as a result of this, computer worms, using a single vulnerability in Windows, have been able to infect a large number of computer systems connected to the Internet. The fact that so many machines on the Internet share vulnerabilities has allowed Internet worms to proliferate and has caused many people to push for diversifying the operating systems market. Evans argues that diversifying an operating system isn't the complete answer. For example, vulnerabilities in widespread network protocols like TCP/IP, applications like the BlackICE firewall, and common libraries like libpng also contribute to this problem. The lesson here is that diversity must be present at several levels of abstraction in order to be effective. Evans cites existing work such as instruction-set randomization and protection of special registers in an address space as examples of diversity techniques in computer systems.

There are limits to the lessons to learn from nature. In particular, Evans notes that computer systems are human-engineered and therefore don't naturally "evolve." In other words, the process of evolution is smarter than we (humans) can be in redesigning our systems, so we shouldn't count on a few iterations of software to evolve into secure software. There are also many instances where nature fails, such as mass extinction of species due to environmental changes and the ability of viral epidemics to wipe out large portions of a species' population. Evans also notes the success of humans in engineering mass destruction in nature, such as vaccines to wipe out smallpox. He uses these points to establish that nature can succumb to "out-of-band" attacks (environmental changes, humans).

In conclusion, Evans says that while there is still much we can learn from biological systems, we have to do much better than nature in designing computer systems, because the attacks that our systems face can also include "out-of-band" attacks that nature often cannot defeat.

FORENSICS AND RESPONSE

Summarized by Zhenkai Liang

Privacy-Preserving Sharing and Correlation of Security Alerts

Patrick Lincoln, Phillip Porras, and Vitaly Shmatikov, SRI

The prevalence of computer viruses and worms drives the need for Internet-scale threat-analysis centers. These centers are data repositories to which pools of volunteer networks contribute security alerts, such as firewall logs or intrusion alerts. However, the possibility of leaking private information in the contributed data prevents people from providing alert-sharing data for those centers. Vitaly Shmatikov presented a practical scheme to provide strong privacy guarantees to those who contribute alerts to these threat-analysis centers.

The authors' solution is an alert-sharing infrastructure, the core of which is a set of repositories where alerts are stored and accessed. In preventing sensitive information (e.g., IP addresses, captured and infected data, system configuration information, defense coverage) from being revealed, several techniques to be used in combination are proposed, including scrubbing sensitive fields, hashing IP address, re-keying by repository, using randomized host list thresholds, and delaying alert publication. To prevent single point of failure, they use multiple alert repositories and randomized alert routing. The alert sanitization techniques protect information in raw alerts, as well as allowing a wide variety of large-scale and cross-organization analyses to be performed. In the performance evaluation, the authors conclude that the cost of their scheme is very low: "a small impact on the performance of alert producers, and virtually no impact on the performance of supported analyses." It is implemented as a Snort plug-in for alert sanitization.

Q: How do you defend flooding of a repository?

A: There's no defense against flooding currently. One possible solution is to let the contributors of alerts register with the system.

Static Disassembly of Obfuscated Binaries

Christopher Kruegel, William Robertson, Fredrik Valeur, and Giovanni Vigna, University of California, Santa Barbara

Christopher Kruegel introduced a static binary analysis technique for disassembling obfuscated binary codes. Disassembly is the process of recovering a symbolic representation of a program's machine code instructions from its binary representations. Recently, a number of techniques have been proposed to make it difficult for a disassembler to extract and comprehend the program's high-level structure while preserving the program's semantics and functionality.

Without the requirement that the code was generated by a well-behaved compiler, their disassembler performs static analysis on Intel x86 binaries. The analysis techniques can be divided into two classes: general techniques and tool-specific techniques. General techniques do not rely on any knowledge of how a particular obfuscator transforms the binary. These techniques first divide the binary into functions. Then they attempt to reconstruct the functions' intra-procedural control flow graph, which is a major challenge of their approach. Finally, a gap completion step improves the result of the previous step. In their tool-specific techniques, the general techniques are modified to deal with binaries transformed with Linn and Debray's obfuscator.

The authors' techniques were evaluated on eight obfuscated SPECint 95 applications. The authors claimed that their disassembler provides a significant improvement over the best disassembler used in the evaluation by Linn and Debray. With the tool-specific techniques, they said that the binary was disassembled almost completely (97% on average). The authors also performed experiments to find the ratio between the number of valid instructions identified by the control flow graph and the number of valid instructions identified by the gap completion phase. In their observation, the control transfer instructions and the resulting control flow graph constitute the skeleton of an analyzed function.

Q: Can you comment how your tool works on encrypted or encoded binaries?

A: The tool is currently a static analysis tool, which cannot help in this situation. It can be extended with dynamic disassembling.

Autograph: Toward Automated, Distributed Worm Signature Detection

Hyang-Ah Kim, Carnegie Mellon University; Brad Karp, Intel Research and Carnegie Mellon University

Hyang-Ah Kim described Autograph, a system that automatically generates signatures for novel Internet worms that propagate using TCP transport. The paper answers the question, “How should one obtain worm content signatures for use in content-based filtering?” The authors noted that all current content-based filtering systems use databases of worm signatures that are manually generated. On detecting a virus, network operators communicate with one another, manually study packet traces to produce a worm signature, and publish the signature to worm signature databases. This process of signature generation is slow in the face of today’s fast-propagating worms.

Autograph takes all traffic crossing an edge network’s DMZ as the input, and outputs a list of worm signatures. A single Autograph monitor’s analysis of traffic is divided into two main stages. First, a suspicious flow selection stage uses heuristics to classify inbound TCP flows as either suspicious or non-suspicious. Then Autograph selects the most frequently occurring byte sequences across the malicious flows as signatures. To do so, it divides each suspicious flow into smaller content blocks using content-based payload partitioning (COPP) and counts the number of suspicious flows in which each content block occurs. The authors also created an extension to Autograph, tattler, for distributed gathering of suspected IP addresses using multiple monitors.

In the evaluation of local signature detection, the authors conclude that, as content-block size decreases, the likelihood that Autograph will detect commonality across suspicious flows increases, so it generates progressively more sensitive but less specific signatures. In the evaluation of distributed signature detection, the authors conclude that multiple independent Autograph monitors clearly detect worm signatures faster.

Q: Do all possible common signatures fall into a single-block content signature?

A: There is a possibility that we miss signatures contained in multiple packets.

Q: Do you look at multiple content blocks for signatures?

A: We are exploring this.

Q: How do you deal with the situation where someone is trying to overload (flood) the monitors?

A: It can happen. The solution is to use distributed monitors to verify signatures.

Nuclear Weapons, Permissive Action Links, and the History of Public Key Cryptography

Steve Bellovin, AT&T Labs—Research

Summarized by Tara Whalen

Steve Bellovin gave the audience an entertaining and informative trip through nuclear weapons technology, Cold War politics, and military history, with a side trip to visit Dr. Strangelove. He provided a thorough discussion of permissive action links (PALs), which were designed to prevent unauthorized use of nuclear weapons. Appropriately enough for a security forum, he proposed an intriguing theory that public key cryptography may have been developed initially as a necessary component of nuclear weapon security.

Historically, permissive action links were required to protect nuclear bombs, not from unauthorized use by enemies or rogue US troops, but to keep them from being misused by US allies. In the 1950s the US was concerned that the Soviet Union was going to invade Western Europe. Keeping a large number of US troops in Europe was not acceptable. The alternative was chosen to create NATO and provide nuclear weapons for NATO allies. However, because of political instability in some European countries and fear that the US could not tightly control deployment of these remote weapons, PALs were added to provide some measure of security.

Out of intellectual interest, Bellovin began independent research into how PALs worked; the documents he obtained from government agencies gave some hints but no conclusive information. The safety features of nuclear bombs were reasonably well documented. These features included a strong link/weak link pair: the strong link provided electrical isolation, while the weak link was designed to fail under stress (such as in an accident). One component of the strong link was the unique-signal generator, a safety system designed to produce a 24-bit signal that was highly unlikely to occur by accident (meant to indicate human intention to deploy). The unique-signal discriminator is designed to lock up if it receives a single erroneous bit.

The PALs themselves were originally electromechanical controls which required two cooperating parties to unlock the weapon. PALs respond to a variety of codes (recently, 6- or 12-digit codes); the weapon is designed to deactivate if too many incorrect codes are entered. Bellovin speculated on design principles that may have been used for PALs. One hypothetical design involves encrypted signal paths: PALs use switches to control the voltage path to the detonators. The original designs used rotors, similar to WWII-era encryption machines, but more recent models use a microprocessor to control the switches; possibly this works by providing an encryption key for the weapon’s environmental sensors, with the signal being decrypted by the strong links. Another hypothetical design involves encrypted code, in this way: It is highly likely that the internal control and sequencing requirements are complex; this timing information, or perhaps the code paths, is encrypted. The public interface could be used for re-keying plus decryption.

Bellovin brought the discussion back to cryptographic history, stating that he found no requirement for the use of public key crypto in PALs (or any indication it was used there, except in one prototype). The PAL sequences are short, which indicates that no secure public key cryptosystem could have been supported. However, consider that by law US nuclear weapons can only be deployed by order of the President; to support this requirement for authenticating the arming code, it is possible that the NSA invented digital signatures. (Added Bellovin wryly, "Including an X.509 naming format for bomb certificates.") In addition, codes need to be put into the PAL: How is this done securely? Since this information should not be passed in the clear, this requirement may have led to the NSA's development of public key cryptography. (This theory is not well documented, but makes for interesting speculation.)

What lessons can security professionals learn from nuclear weapons safety and security designs? One important lesson is to understand exactly what problem you're solving: The weapons designers got it right because they knew precisely what the real problem was. For access control, you want to find the One True Path that everything must pass through, and block that.

For more information on this topic, see Steve Bellovin's PAL Web page at http://www.research.att.com/~smb/nsam-160/pal.html.

DATA PRIVACY

Summarized by Rachel Greenstadt

Fairplay—A Secure Two-Party Computation System

■ *Awarded Best Student Paper!*

Dahlia Malkhi, Noam Nisan, and Yaron Sella, Hebrew University; Benny Pinkas, HP Labs

Yaron Sella presented an implementation of Secure Function Evaluation (SFE), a theoretical construct studied in the cryptography community for 20 years. The classic example of SFE is the millionaire's problem. Two millionaires want to know who is richer. One has \$X million and one has \$Y million, and they don't trust each other. They need either a trusted third party or SFE. In this case the function is > = <, but it could be any function. The goal of the research was to see whether two-party SFE was practical and to better understand where the bottlenecks were, in computation or communication. They wanted to push this protocol from the theoretician's realm to the practical arena.

They defined two languages, Secure Function Definition Language (SFDL) for specifying the function to be evaluated, and Secure Hardware Definition Language (SHDL) for turning the program into a circuit. Thus if Alice and Bob want to evaluate a circuit, Bob takes the circuit and garbles it *m* times and sends these to Alice. Alice chooses one and tells Bob which. Bob then reveals the secrets of all the non-chosen ones and Alice verifies that they are okay. This is the commonly used cut-and-choose method of indirect randomized verification. Bob types in his input and sends the garbled version and Alice types her input. They engage in oblivious transfer where Bob is the sender and Alice is the chooser. Alice evaluates, sends to Bob, and they both print the results.

Yaron then gave the audience a live demo of Fairplay. He first showed examples of SFDL and SHDL (the millionaire's problem) and created circuits. He then showed examples of running the billionaire's problem (bigger numbers). There was a short delay as the circuits were sent; Bob had 100000, Alice 100001. So Bob received 0 and Alice 1. The code was written in Java, using its crypto-libraries and SHA1. They tried several functions: AND, billionaire's problem, keyed database search, and finding the median of unified arrays. The main bottlenecks were in the communication of the circuits and the oblivious transfer. You can play with the system at http://www.cs.huji.ac.il/danss/Fairplay. An audience member suggested that the delay experienced on the LAN might be reduced by turning off nagling.

Tor: The Second-Generation Onion Router

Roger Dingledine and Nick Mathewson, The Free Haven Project; Paul Syverson, Naval Research Lab

Roger Dingledine began by explaining that TOR (The Onion Routing) was similar to Zero Knowledge's Freedom networks, except with some improvements and free. It aims to provide anonymity by protecting packets from traffic analysis. The research is funded by the government (they want to look at Web sites without revealing their .gov and .mil addresses and to protect their networks from insider attack), but anonymity is also useful for individuals, corporations, and even law enforcement. People are worried about criminals, but they already have anonymity (Nick Weaver can tell you more about this). The idea behind the network is to distribute trust among several nodes and that not all of them will collude and do traffic analysis on the users. The system provides location anonymity, not data anonymity. You should use something on top of it to scrub your cookies and whatnot.

Onion Routing is an overlay network of anonymizing proxies using TLS between each pair of links. Alice, the client, connects to the first hop, negotiates a key, then tunnels to the second hop, and then to the third hop. Each node only knows its predecessor and successor. Once built, you can multiplex TCP streams over these circuits, which are rotated periodically. To connect, you contact the directory servers (more trusted nodes) and they tell you about who's in the network, keys, exit policies, etc. There's a human in the loop to prevent Sybil attacks. They have 50 servers. TOR is 26,000 lines of C code that runs in user-space. The client looks like a socks proxy, so they don't need to build an application for each protocol they want to support. There are flexible exit policies to deal with abuse and legal issues. The system works with only a couple of seconds of latency. The system also supports location-hidden services, so Bob can run an Apache without revealing his IP. At http://freehaven.net/tor there are specifications, a second implementation of the TOR client, a design paper (this one), and code.

People asked about abuse issues, and Roger answered that there haven't been any, but they are a research network and they block SMTP. Someone asked whether someone could follow the packets by comparing their sizes. Roger answered that they already give up if the adversary can see both sides of the connection; their goal is to prevent all attacks that are easier than that. Someone wondered whether you should send the hostname or the IP address through TOR. The answer is, the hostname; the resolve has to be done at the end or DNS (since it is UDP) will break anonymity.

Understanding Data Lifetime via Whole System Simulation

Awarded Best Paper!

Jim Chow, Ben Pfaff, Tal Garfinkel, Kevin Christopher, and Mendel Rosenblum, Stanford University

Jim Chow began by laying out a common scenario: a user at a password prompt. The question was, after you type your password, what happens to this sensitive data? Under the assumption that the software was written with security in mind, it will read the password into a buffer locked in memory, use encrypted software, and erase the password once it's done with it.

If your computer is broken into at a later date, that sensitive information is assumed by most people to be gone, but this paper showed that not to be the case. Even if the application has been careful, many extra copies of the data often exist in the tty buffer, the X server, gtk+ buffers, etc. The problem with data lifetime issues is that you can only erase the copies you know about; the others may get written to disk.

To address this problem, they wrote TaintBochs, a program to do whole system simulation and track sensitive data as it flows through software. However, just figuring out what was a copy was nontrivial; where this was the case, the program erred on the side of caution.

They learned that there were usually several copies of data stored in the kernel, and although kernel memory isn't paged, there are plenty of ways to dump it to disk, from hibernating laptops to crashes. Since indirection is so popular, it's impossible to avoid middlemen processes, so you get a lot of copies there, too. Fixing things is usually easy, though: Zeroing and string destructors destroy most taints, and TaintBochs can help with that.

During Q&A, Jim pointed out that there are possibly false negatives in the experiments, since TaintBochs can only prove the existence of taints, not prove you don't have problems. Someone else brought up the point that sometimes the compiler doesn't obey the `memset` command, but there are lots of ways to get around that problem.

My Dad's Computer, Microsoft, and the Future of Internet Security

Bill Cheswick, Lumeta Corp.

Summarized by Serge Egelman

The biggest threat to Internet security is not large corporate computers that are specifically targeted by intruders, but individual home machines that are simply configured poorly. Bill Cheswick elaborated on this by using his father as the typical user. He's an 80-year-old veteran who runs Windows XP. He uses the computer to trade stocks, check email with Microsoft Outlook, and chat with family and friends via AOL Instant Messenger. He has been "skinny dipping" for years, by which Cheswick means that his father does not have a firewall or any spyware protection, and he never updates his anti-virus definitions. Pop-ups appear on the screen every three minutes or so, but his father simply dismisses them without giving any thought to it. In fact, a computer repair person came over to take a look at the machine and discovered a variety of spyware programs, redirections to obscene Web sites, and viruses that he had never heard of before.

The question is, why should he care? He found a way of getting around all of these distractions so that he could still be productive, and he did not want a system administrator changing any of his personal settings. Since malicious attacks are very rare, he did not see any real reason to bother changing any of his habits. On the other hand, his machine had become a slave that could be used in propagating spam or denial of service attacks across the Internet, thus disrupting other people's work. Cheswick described it as a "toxic waste dump spewing blue smoke across the Internet." It is in everyone's best interest that these machines are secured.

Computer security has become an arms race. Originally anti-virus software worked by looking for specific signatures in the virus code, but this was later thwarted by viruses that would encrypt or recompile themselves. So the detectors started running simulations to see how the virus would act on the program, but now newer viruses can detect these simulations. Trusted computing offers a solution to this problem, but it is only a matter of time before the virus writers regain control. Similar arms races can be seen in password sniffing, password cracking, hardware vulnerabilities, and even software upgrades. In an homage to The Karate Kid, Cheswick quotes Mr. Miyagi: "The best block is not to be there." That is, the key is to stay out of the game altogether.

In order to keep Dad out of the game, better client security is needed. Windows ME has eight ports open by default, and Windows 2000 and XP have increasingly more. A similar machine running FreeBSD will have one open for SSH, if any. It is obvious that these ports should not be open by default. Personal firewalls try to solve this problem, but additional software really is not needed here; the problems are caused by all the programs that are not used.

Cheswick has a solution, though: Windows OK. This could be a new end-user-oriented operating system that functions as a thin client. No firewall or anti-virus software will be necessary, since no ports will be open by default. Additionally, all programs will be signed by trusted parties, and all security settings will be located in one central location. Additionally, macros will not function in Word or PowerPoint and can only be used for arithmetic in Excel.

Such a system would be immune to the vast majority of security threats on the Internet. Cheswick believes that we may never win the malware detection arms race, so instead we must win the protection game. Unfortunately, most users simply do not see the problem; a fully hosed computer may still seem functional to them. What the user does not know is that his system is affecting other people. By creating a usable system with tight security by default, we can keep the average user out of the security game altogether.

For WiPs and Posters, please see http://www.usenix.org/events/sec04/tech/wips/.

For audio files of some sessions, see http://www.usenix.org/publications/library/proceedings/sec04/tech/mp3/.

NTT DoCoMo

DoCoMo USA Labs

Be part of the future

DoCoMo Communications Laboratories USA, Inc.

DoCoMo USA Labs – The leader in Mobile Software Research and Development

Future generations of wireless networks will provide virtually unlimited opportunities to the global, connected community. Ensuring maximum user benefit from the future environment requires that the services and application technologies are implemented with truly exception vision and imagination. The realization of these new technologies is our mission. DoCoMo USA Labs was formed in 1999 to develop applications and services for the fourth generation of mobile services and beyond. With clear vision, aggressive innovation, constant experimentation, and carefully planned growth we have established a global centre for excellence where we are working in a collaborative environment creating the future of mobile communications.

DoCoMo USA Labs is part of the world leading NTT DoCoMo group of communications companies, the premier wireless service provider in Japan and a leader in innovation throughout the world of mobile communications. We are located in Silicon valley where we are conducting research in advanced mobile software, wireless applications, architectures, networks and protocols. DoCoMo USA Labs has a particularly strong and growing focus in mobile software.

The Mobile Software Laboratory

The mobile software laboratory investigates software structures for handsets and servers for 4G generation networks. The goals of our research are to enable the creation of new applications, the easy deployment of new applications and the maintenance of network nodes and end user devices. Our current generation of services is based on i-mode and i-appli infrastructures. These have been very successful in drawing new content and applications for our customers. Our view of next generation networks is that they are "dynamic" and "open." The networks are open in that third parties can participate in our networks and devices more easily and, dynamic in that new features can be added to networks and devices after deployment since it is impossible for us to predict all the features that are required by our networks and devices at deployment time. One important aspect of the new networks will be the enabling software that runs on the networks and devices. In particular, we focus on three key areas: operating systems, middleware and device security.

Operating systems:

Our phones for open and dynamic 4G networks will run multiple applications simultaneously, run complex applications such as multiplayer games, run multi media applications all in a single operating system platform. One of the most important goals of an operating system is to mediate applications access to hardware. To enable better mediation of hardware we are focusing on several important areas including, but not limited to, energy management, advanced protection domain structures for processors for cellular phones, efficient process scheduling techniques, efficient data storage management techniques, and self managing systems.

Middleware:

Mobile middleware focuses on providing intelligent support for applications on devices and for enabling intelligent communication between devices and the network nodes. For example, mobile user often experience periods of disconnection and an intelligent middleware layer and mask brief periods of disconnection. Our research into middleware focuses on efficient communication management, efficient reconfiguration of applications and systems based on system conditions, and efficient repartitioning techniques for applications.

Device security:

The device security project focuses on new techniques for securing the device. We are investigating new language based techniques for enabling safe downloads of software at all levels of the software stack, new techniques for secure dynamic reconfiguration of software, and understanding and developing methods to prevent attacks on devices through various channels including new types of network interfaces including Bluetooth, iRDA, WiFi and WiMAX.

—Vice President & Lab Director, Nayeem Islam, Ph.D

Visit www.docomolabs-usa.com

JOIN US IN ANAHEIM IN 2005
for the latest groundbreaking information
on a wide variety of technologies and environments.

USENIX '05

Annual Technical Conference

April 10–15, 2005
Anaheim, CA

Check out the Web site for more information! www.usenix.org/usenix05

;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