ABE SINGER

tempting fate

Abe Singer has been a computer security researcher with the Security Technologies Group at the San Diego Supercomputer Center for the past five years. His work has involved growing SDSC logging infrastructure and analysis capabilities, participating in incident response and investigation, and working with the TeraGrid Security Working Group. Mr. Singer, with Tina Bird, is the author of *Building a Logging Infrastructure,* SAGE Short Topics booklet #12. Mr. Singer's current research is in automation of syslog parsing and analysis toward data mining of logs for security. In addition to his work at SDSC, Mr. Singer is an occasional consultant, expert witness, and lecturer. Prior to SDSC, he was a consultant for several years and a programmer and system administrator for over 15 years.

abe@sdsc.edu

IN THE DECEMBER 2003 ISSUE OF *;login:* I wrote an article called "Life Without Firewalls" in which I talked about how we do security at SDSC, why we do not use firewalls, and how we have been very successful at keeping out intruders.

If I were superstitious, I'd say I should have known better than to tempt fate. In the Spring of 2004, SDSC had an intrusion that gave us a pretty good amount of grief.

Rik Farrow suggested that I call this article "Eating Crow," but I stand by what I said in my previous article. The intrusion was successful only because we didn't follow our own rules well enough. Our strategy helped us detect the intruder quickly and reduced the scope of the intrusion (it could have been much worse, and for some other sites it was). Eight of our hosts (out of several hundred) had root compromises; moreover, the intruder was able to modify user-owned files on one of our NFS servers. Our reference system model allowed us to have the compromised hosts reinstalled and up and running in less than two hours each—we didn't have to think twice about reinstalling a host.

As for our lack of firewalls, in this case a firewall would not really have helped (as I'll explain below). In fact, shortly after the attacks, Marcus Ranum sent me an email saying simply, "Living without Firewalls . . . ;-)," so I explained what happened, to which he responded, "aw crap, transitive trust, gets 'em every time."

So I'll explain how our intruder got it, where we failed, and how our security strategy helped mitigate the problem; I'll also talk about what we've learned and what we're doing differently. I'm going to be deliberately vague about some things, to protect the privacy of some of the people who have been compromised and because the intruder is still actively attacking sites.

Beginning in December 2004, we started hearing about compromises at other sites. The intruder had gotten root on some machines and had installed a trojaned SSH client, which he used to gather usernames and passwords to other sites as users logged on to the compromised hosts and then SSHed into remote sites.[1]

The intruder would then log in to a user's account at the remote site and look around for ways to compromise the host or any other host at the remote site.

By March 2004, we knew of several sites that had compromises. Tina Bird published a bulletin which described the activity at Stanford and elsewhere.[2]

One morning in late March I received an email from one of our systems about a failed sudo attempt by one of our sysadmins (our version of sudo sends email to root when it fails). A quick phone call verified that it indeed was not the sysadmin. The host on which this happened was the host we use to manage the rest of our machines—those machines allow root rsh from the management host so that we can automate configuration of multiple hosts (this was described in my previous article). Of course, this definitely got our attention.

A quick check of logs found an rlogin[3] to that account from a workstation. A ps on that workstation showed a root-owned process called "foosh"—definitely a bad sign.

The process disappeared within minutes of our looking at the host. We believe the intruder had spotted us and decided to leave.

A check of the logs showed a particular user logging in (via SSH) to one of our workstations from a remote site with which we collaborate, and within a minute that user logged in again from a cable modem somewhere in the Pacific Northwest. Following immediately were logins from the workstation to every other system on which the user had an account (and numerous failed attempts on hosts where the user did not have an account).

Our logs also showed, shortly after the rash of logins, that root was su'ing to several users, including the user who had initially gotten my attention. The tty from which the su's were executed corresponded to one that the suspected user had logged in to, and that user didn't have any privileged access. This was our indication that the intruder had definitely gotten root on a host.

Various log entries gave us a clue about what the attacker was doing. He had managed to get root on a system and had done some investigating to determine who might have privileged access. We think he looked at things like who was in the root group, the SSH known_hosts file, etc. He targeted our management system, which only has accounts for those users who need it. Thus, he needed an account on that host for which he didn't already have the password. He picked a user who had access and su'ed to that user in order to be able to rlogin to the management system. Once there, the intruder apparently tried sudo, hoping to take advantage of cached sudo credentials, which failed.

So how did the intruder get root on the first host? We're pretty sure he used a local kernel exploit. We had patched the host but had not rebooted it, so the patch had not actually taken effect (Mistake #1). We figured this not only from our knowledge of the patch state of the host, but because the intruder placed an executable in a user's home directory and modified the user's .cshrc to try to run the binary anytime that user logged in to a Solaris box. The intruder was trying to get users to root boxes for him. Fortunately, the intruder wasn't very good at writing shell scripts, as there was a syntax error in the .cshrc file.

The logs also showed the intruder trying things like putting a "+" in .rhosts (which is disabled on our hosts and is logged when an rlogin is attempted). We found keys added to various users' .ssh/authorized_hosts file. Process accounting also showed the intruder running a program called "n," but he had erased his tools before logging out, so at the time we didn't know what it was.

We rebuilt the machine and rebooted others that had the same patch applied. We checked all users' authorized_hosts and .rhosts files to make sure there were no other accounts accessible. We changed the password for the known compromised account.

I also called the site from which the compromised user had originally logged in—where the password had been intercepted—to let them know they probably had a compromise. They called me back a couple of hours later and confirmed that they had been owned.

A couple of days later, we found some modified authorized_keys files again and discovered that another Solaris host was compromised. This host was not vulnerable to the same exploit that had been previously used, so the intruder had another exploit. This host had also been patched but not rebooted. We did a lather, rinse, repeat—reinstalled and rebooted hosts that needed it. This host was also a Solaris 8 host, as was the first host compromised, so we decided that all Solaris 8 hosts needed to be patched and rebooted (Mistake #2).

That weekend we discovered the intruder had gotten root on a Solaris 9 box. We then realized we needed to make sure all of our hosts were fully patched and rebooted. Machines were carefully rebooted one by one to make sure they came up okay, and at midnight a few hosts that ran special applications were left for the sysadmins who administered them to reboot (Mistake #3).

The next morning, before the machines had been rebooted, the intruder got root on another host, su'ed to yet another user, and sent an email out to every email address at SDSC and UCSD that he could find, with some rather rude ASCII art and some typical script-kiddie language, talking about how great he was and what losers we were.

That was embarrassing, but we were able to recover, reinstall the host that was compromised, reboot everything, and make sure all our patches were up to snuff.

After that, we did not discover any more root compromises on our Solaris or Linux hosts (I'll qualify that with "that we could detect"). News of the various intrusions spread, and we started getting calls from the press. A spokesperson was appointed to deal with the press. He was quoted as saying that the attacker had only gotten a few perimeter systems and had not gotten at our infrastructure (Mistake #4), which was true at the time: A few workstations had been compromised, but our file servers were intact, and we had no indication that the intruder had gotten onto any of those hosts (although we did have some indications that he tried).

The day after the newspaper article came out, the intruder got root on the login node of one of our supercomputers, did a "wall" to all the users with some more ASCII art, and did a "shutdown" on the host. In the "wall" message, he quoted the part of the news story about not getting at our infrastructure and claimed that this shutdown was proof that he *had* actually gotten at our infrastructure. We believe the intruder exploited an unpatched FTP server that shouldn't have been running on the host in the first place (Mistake #5).

So that system was taken offline. It was due to go out of production in a few weeks anyways, so we just left it offline.

Somewhere in the mix of this, I received a call from someone at another university. They had found John-the-Ripper running on a cluster of theirs, with what appeared to be a fragment of our shadow password file, including my encrypted password. He sent me a copy, and I was able to confirm that it was indeed my password.

Thus, we also knew that the intruder was cracking passwords in addition to running trojaned SSH clients at other sites.

When we took down the supercomputer, we also changed passwords for all users (several thousand) and audited our other supercomputers to make sure that there weren't signs of a compromise.

From then on, things calmed down. We continued to see the intruder log in to compromised accounts (even with the password changes), but no sign of root compromise. This was an *acceptable* state, not a great state, but we could live with it; our big concern was root compromise and compromise of our infrastructure—the file servers, DNS servers, and such.

I then received a call from someone at yet another university. He had found what appeared to be a copy of my email inbox. He gave me the header timestamps from the first and last message, and they corresponded to messages in my inbox. And the dates were from a couple of weeks *after* we had cleaned everything up. We had no idea at the time how the intruder had accessed it—there were no signs of root compromise anywhere. And we didn't know how my account could have been compromised, as I had not logged in from other sites—had not left credentials available for use—and the logs did not show any suspicious logins to my account. A log message showed some more failed attempts to use "+" in my .rhosts file, but we didn't know how it had been put there. The assumption was that my password had somehow been compromised. So I changed all my credentials and started logging in only from my laptop using an SSH private key that had never left the laptop. (It's a Mac OS9 laptop, so I felt pretty good about the integrity of the OS.)

Finally, a few weeks later, we figured out how the intruder had done it. We had heard of a well-known tool called nfsshell4 that the intruder had used at some other sites, and so we tried it at our site. It worked. The nfsshell tool exploits NFS servers that allow clients to send mount requests from an unprivileged ("high") source port. And once a file system is mounted, there is no validation of the UID used in NFS file operations. In other words, nfsshell allows an unprivileged user to mount a file system and then read and write files using any UID. Since we squash root access via NFS, the intruder was unable to write files as root, but was able to write to files owned by any other user. So that's how he was able to stick SSH keys into authorized_keys files, and how he got at my inbox, etc.

While we were looking at this problem, but before we could react with a fix, the intruder decided to erase a couple of home directories. At that point we took SDSC off the Internet until we could remedy the situation.

It turns out there was a simple kernel parameter that had to be set to disable unprivileged ports, and that parameter had not been set. nfsshell is a very old exploit, and our previous server had been immune; we had made the assumption that the new one was equally immune (Mistake #6).

We fixed the file server, cleaned up, and brought SDSC back online.

Since then, we continue to have the occasional user account compromise, but no new root compromises. We have seen the attacker come back and try the same exploits, with no success.

Based on our lessons learned, we have changed a few things at SDSC: We have shortened our patch cycle (it was about a month long), and reboots get priority over uptime. We have implemented two-factor (token-based) authentication on our critical infrastructure machines, as the only people who have to access those are system administrators.[5]

So now you're probably wondering why I stand by my previous article and why I say a firewall would not have helped. If we had had a firewall, we would have had to

allow inbound SSH, so that our users could log in. The intruder used the same access mechanism our users do, often from the same outside hosts.

Second, as I indicated at the beginning of this article, we mostly were owned because of things we *should* have been doing. It does show, however, that doing it right is *hard*. You have to, well, do *all* of it right. If we'd had our machines fully patched and rebooted, the intruder probably wouldn't have gotten root, at least not with the techniques that we saw him use.

Also, many of the things we do mitigated the extent of the compromise. For instance, squashing root and set on our file servers kept the intruder from creating setuid-root files and running them on other hosts.

Having Kerberized sudo kept the intruder from getting sudo privileges, even when he managed to obtain a user's password. Having a separate set of credentials for privileged access is definitely a Good Thing.

Our centralized log server allowed us to look at log activity across hosts and provided assurance that we had a good copy of log information, even if the intruder erased or altered logs on the compromised host.

And having reference systems allowed us to recover quickly. Our Solaris hosts take a couple of hours to reinstall, mostly unattended (it takes a while for a new host to install all of its patches). Our Linux hosts take around half an hour.

We fared better than some other sites. For instance, we know of one site that allowed root logins via SSH for sysadmin purposes. The intruder owned the workstations that the intruders were using to manage other systems, and from there just got the root password. Another site that we know of did not have root squashed on their NFS server, so the intruder was able to create setuid-root programs from one host and then log on to another host and run the program. Yet another site had to rebuild all of their compromised hosts by hand—they had not automated the process.

Some lessons we learned about intrusions: Never underestimate the capabilities of your attacker. Assume your communications are being monitored. Don't taunt the animals. And reboot everything.

REFERENCES

1. In the academic community this is a very common practice; organizations are involved in collaborative activities, researchers may be at one institution and be using the facilities at another, etc. In fact, most of SDSC's several thousand users are located at other institutions, as our business is to provide computing resources to researchers.

2. Stanford University Information Technology Systems and Services, "Security Bulletin on Multiple UNIX Compromises," http://securecomputing.stanford.edu/alerts/multiple-unix-6apr2004.html.

3. I often get funny looks when I mention using rlogin. We *only* allow r-commands between hosts that we manage, and we only allow managed hosts on the networks that this traffic travels over. So spoofing tcp connections requires access to the local network.

4. nfsshell, ftp://ftp.cs.vu.nl/pub/leendert/nfsshell.tar.gz.

5. I still believe that token-based authentication is too much of an expense (not just the hardware costs, but the support overhead) to do for all users, but it's appropriate for those who have privileges.