Insider Tips: Tracing Users

Who is Who?

Admins need to be able to trace what user logged onto the system when and from where. This allows you to protect your computers from abuse and trace intrusions should the worst come to the worst. BY MARC ANDRÉ SELIG

Modern computer networks are an extremely practical thing. When I need to manage the cluster at the university remotely, I can log on to the network and install a Sendmail update without even leaving my couch. A computer in Pennsylvania takes care of filtering spam for me, and of course I can just as easily program that computer from my home.

Unfortunately, malevolent hackers enjoy the same kind of freedom. Once a hacker has the credentials for a Unix system – in the form of a user account and matching password, or a cryptographic key – he or she can typically log on from any location. And as long as Unix administrators insist on using (or playing games on) Windows machines, there is always a danger of worms and trojans grabbing credentials.

Virtual private networks help restrict the attackers, but VPNs are unwieldy and hard to set up, and that rules them out for many small businesses and home users. Admins thus check who has been using their machines, and where these logins come from. This is the only way to detect irregularities and chase off intruders.

Who is Online?

The *who* command (see Figure 1) provides a quick overview of the current users. Besides the username and origin, the command also outputs the date and time of the login and the (virtual) console where the user is working. Three people are logged on in our example: *baier* is working at the console, she is running an X server (*:0*) and a whole bunch of other terminals (*pts/X*). We can assume that these are Xterm windows.

```
Terminal <2>
[mas@sun3 mas]$ who
baier    console  Oct 13 07:36   (:0)
wwwadm   pts/1    Nov  3 07:39   (zpidsu5.uni-trier.de)
wwwadm   pts/2    Nov  3 07:40   (zpidsu5.uni-trier.de)
wwwadm   pts/3    Nov 10 10:11   (zpidsu5.uni-trier.de)
baier    pts/4    Nov  3 07:36   (:0.0)
baier    pts/5    Nov  3 07:36   (:0.0)
baier    pts/6    Nov  3 07:37   (:0.0)
baier    pts/7    Nov 10 10:08   (:0.0)
baier    pts/8    Nov 10 10:08   (:0.0)
mas      pts/9    Nov 10 20:09   (acb6ae4b.ipt.aol.com)
[mas@sun3 mas]$
```

Figure 1: The *who* command outputs a list with the users currently logged on to the system. Baier is running an X server on the console (*:0*), while *wwwadm* and *mas* have logged on remotely

```
Terminal <2>
mas@ishi:/export/home/mas> w
 20:13:50 up  2:09,  3 users,  load average: 0.00, 0.01, 0.00
USER     TTY      LOGIN@   IDLE   JCPU   PCPU WHAT
mas      :0       18:07   ?xdm?   1:58   0.00s -:0
mas      pts/0    18:07    2:06m  0.00s  0.28s kdeinit: kwrited
mas      pts/1    18:07    0.00s  2.05s  0.00s w
mas@ishi:/export/home/mas>
```

Figure 2: The *w* command on Linux provides more detail than *who*, also showing the uptime and load for the computer

```
Terminal <2>
[mas@sun8 mas]$ last -5 mas
mas      pts/1        acb53cfc.ipt.aol Mon Nov 10 20:54   still logged in
mas      pts/3        acb0c055.ipt.aol Sun Nov  9 21:12 - 21:12  (00:00)
mas      pts/1        pd90249e0.dip.t- Tue Oct 14 20:15 - 20:38  (00:22)
mas      pts/1        pd9024626.dip.t- Tue Sep 30 18:54 - 18:54  (00:00)
mas      pts/1        pd902475d.dip.t- Tue Sep 30 13:38 - 14:50  (01:11)
[mas@sun8 mas]$
```

Figure 3: If you are interested in the history, a call to *last* will show a list of logins with their origin and duration

In addition to *baier*, there is an account called *wwwadm* which logged on via a host called *zpidsu5.uni-trier.de*, and *mas* via *acb6ae4b.ipt.aol.com*; this is obviously a dial-up line.

How long here?

The *w* command (see Figure 2) is a close relative to who, and available on most Linux systems. Besides being shorter to type, *w* also displays the uptime, and the current command line for each PTY (pseudo terminal).

Both of these traditional tools, *who* and *w*, display only the active users. The *last* command (see Figure 3) is a logical extension of this, as it shows the most recent logins. The command searches back entries to the point where accounting was initialized on the system and can easily return thousands of entries.

It would be difficult to manage mass entries on a heavily used system. But thankfully, *last* provides two useful filters: for one thing, you can restrict the list to a single username, or a single terminal (such as *tty1* for the Linux console) in some cases. For another, a parameter such as *-20* tells *last* to restrict the output to the twenty

(latest) login entries. These filters can be combined: *last -5 mas* will list the last five logins for the user *mas* (see Figure 3).

who and *last* reference special protocol files to obtain this information. The file *utmp* records the active users, and is stored in */var/run* on modern Linux systems. The system uses *wtmp* to log login and logout data; this file is typically in */var/log*. But the exact position of both files will depend on the philosophy and the age of the Linux distribution; */var/spool*, */var/adm* and */etc* are typical locations.

How it works

wtmp contains a long list of logons and logouts (in binary format). The data entry is created by *init*, *agetty*, or *login* for console logins, and *init* records logouts, reboots etc.

In contrast to this, *utmp* contains exactly one entry for each user. The entry is a record of the last login. Depending on how up-to-date the C library is, *utmp* may look fairly large: traditionally, the file is long enough to contain an entry for every possible user. That is quite large, considering that over 65,000 users are permitted. To prevent all this space going to waste, the file is sparse – the unused areas are simply padded with null bytes, and not stored on the hard disk. This trick also accelerates access to the file.

Programs should not access *wtmp* and *utmp* directly. Well-behaved programs will use the library functions *utmpname()*, *setutent()*, *getutent()*, and *getutid()* instead. Incidentally, some Unix variants use *wtmpx* and *utmpx*, rather than *wtmp* and *utmp*, to store the data in an extended format. This does not make sense in Linux' case as the original file formats fulfill the requirements placed on them.

Access privileges are typically organized to allow any user to read the databases, while write access is restricted to specific processes. Write access is assigned to *init* for logouts, and runlevel changes (these include the boot process and shutdowns), the *getty* processes and *sshd* for hardware-based and virtual TTYs, *login* for successful login attempts, *sessreg* for GUI-based logins that use *xdm* and the like, and GUI-based terminals such as *xterm* for many environments.

It is typically unnecessary to define complex user groups to provide access controls for *utmp* and *wtmp*. The login program accesses the database before dropping its root privileges and assuming the privileges of the user logging on.

More Logfiles

The databases mentioned so far give information on numerous activities that take place on a system. They assume processes will be conventional and update the files as expected.

Also, the data must comply with a precisely defined structure. The maximum length of the hostname used for logging on is typically tightly restricted. Abbreviated hostnames, such as *pd90249e0.dip.t-* in Figure 3 are okay, but useless for incident response or forensics.

Sometimes you really do need more information – particularly in the case of complex login procedures that use cryptographic authentication, where detailed error reporting is essential. This kind of information typically ends up in a central logfile, referred to as the syslog. We will be looking at the syslog next month.