Insider Tips: Identd with Linux-based Servers

NAME TRACER

In last month's issue of Admin Workshop, we introduced tools that help admins get services up and running. This month, we will show how you can use the Ident protocol to associate a user name with a TCP connection. **BY MARC ANDRÉ SELIG**

In follow-up to last month's discussion of inetd-based server processes [1], this month's issue examines the Ident protocol as an example of the possibilities and pitfalls of Unix-based servers. The very simple Ident protocol serves the purpose of assigning a user name on a client machine to a TCP connection.

FTP, IRC, and SMTP often rely on Ident. When an FTP connection is established (and depending on the server configuration), some FTP servers ask the client to identify the local user who opened the connection. Figure 1 depicts this process.

The FTP server itself then becomes an Ident protocol client. It opens a connection to port 113 on the requesting machine. As only root processes are allowed to bind to this port (because the port is in the privileged port range below 1024), the response can be trusted to provide somewhat accurate information.

The protocol itself is extremely simple: the Ident client code sends the two port numbers for the FTP connection to the Ident server (in Figure 1, the ports are 33812 and 21). The Ident daemon then checks the system to find out which user owns the process bound to port 33812, and returns the ID to the client. Some daemons simply return a hash instead of the user ID.

Ident Security

Ident is a legacy application built when most Unix computers were multiuser systems that typically lived in computer rooms and were managed by one or more administrators. If a user misused the computer and attacked an Internet server, for example, the administrators had an easy way of detecting the user behind the attacks.

Installing an Ident daemon allowed admins to track use. The operator of the server that had fallen victim to the attack would simply call the admin at the computer center where the attack had originated and let that admin know which user ID had been logged. This allowed the local administrator to check out the account and quickly discover the evil-doer.

Of course, the Ident service only helps in straightforward cases. Attackers with a modicum of skill could easily find ways to undermine the defenses. Thus, Ident is no protection to the server whatsoever, particularly considering the fact that the protocol does not support authentication or authorization.

Common Issues

Ident can definitely mean a security risk for clients. A user connecting to an untrusted server would give that server a lot of additional information about the client machine. And an attacker could misuse this information to attack the client. To combat this, some Ident imple-

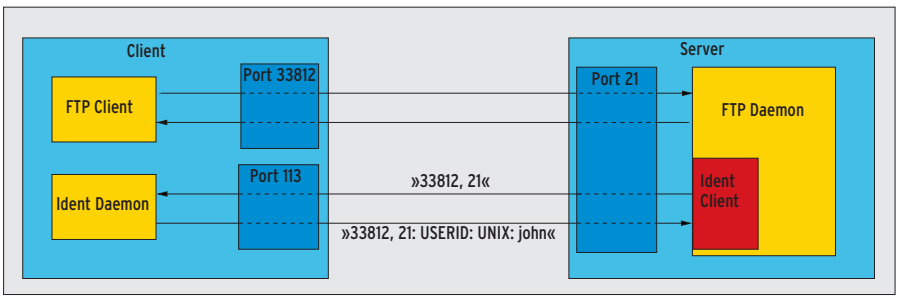

Figure 1: When a client opens a connection, the FTP server asks the client to identify the user requesting that connection.

mentations send a hash (made up of the encrypted user name and a time stamp) instead of the ID and simply note the genuine user name in a log file. In this case, the remote server never gets to see the credentials.

As Ident dates back to the era of centrally managed host machines, its usefulness is restricted today. Most Internet users have their own computers and are their own administrators. Users can get their Ident daemons to respond with anything they like. In other words, server operators should never trust Ident daemon responses, let alone use them for authentication purposes.

Let me emphasize that Ident is not a good tool for hardening machines, but it can be a forensic tool. Many Internet servers (especially IRC and SMTP daemons) require Ident to be running client-side. If a user has not launched the service client-side, a delay can occur on opening the connection. This delay is caused by the client rejecting the incoming Ident requests and the server waiting for the response to timeout.

So it does make sense to install Ident on your machine. Let's look at the daemon setup to illustrate a few common scenarios that occur when setting up a server.

Setting up Ident

Most Linux distributions have at least one Ident implementation. As the service is not commonly used, it makes sense to use either the inetd or Xinetd [1] super server to launch Ident. If you opt for the variant that uses the legacy inetd super server, you need to add the following to */etc/inetd.conf*:

```
ident stream tcp nowait nobody➥
 /usr/sbin/in.identd
```

If your distribution supports Xinetd, you need to create a file called */etc/xinetd.d/identd* and add the lines shown in Listing 1. Some distributions have a sample configuration, which you just need to enable. To do so in inetd, simply remove the # signs at the start of the appropriate lines. In Xinetd, you need to change the *disable = yes* line for a service to *disable = no* to enable that service.

After changing the configuration, the admin needs to send a HUP signal to tell the Internet super server to parse the new configuration. (*killall -HUP inetd* or *killall -HUP xinetd*).

Figure 2: You can configure your router to pass Ident-related data.

Setting up Packet Filters

Setting up and launching a daemon may not be enough. Many of today's desktops now have either external or local firewall protection. A correctly configured packet filter will block all Ident packages. This blocking of Ident packets also explains the delays when connecting to some Internet servers, as the packet filter simply removes any incoming packets destined for port 113. Most servers will not wait for the response packets for long, but this can still cause a delay of a few seconds when opening a connection.

The way round this is to drill a hole in your firewall. The IPTables tool on Linux 2.4 or later allows you to set up rules to accept incoming packets from the Ident server and send the matching responses. Working as root, simply type the following lines to punch a couple of holes in your packet filter:

Listing 1: Xinetd Entry for identd

```
01 # file /etc/xinetd.d/identd
02 service ident
03 {
04     socket_type = stream
05     protocol    = tcp
06     wait        = no
07     user        = nobody
08     server      = /usr/sbin/in.
   identd
09     disable     = no
10 }
```

```
iptables -I INPUT -p ➥
tcp --dport  ➥
ident -j ACCEPT
iptables -I OUTPUT -p ➥
tcp --sport ➥
ident -j ACCEPT
```

Access Routers

The configuration we have looked at so far works fine, assuming the computer connects directly to the Internet using a modem, ISDN, or DSL. If you have a router that provides the connection and routes packets onto the Internet, you will need to modify the firewall rules on your router to handle Ident (see Figure 2). How you do this depends on the kind of router you have.

Hurdling NAT

You may be wondering what happens if your network uses Network Address Translation (NAT) to share a single IP address among a number of clients.

A simple packet filter will not work. Incoming Ident requests always use a reserved port 113. You would need an intelligent firewall that remembers which computer has opened a connection to a remote IP address to assign the Ident packets correctly.

To avoid this confusion, and at the same time avoid delays while opening FTP, IRC, or other connections, configure your packet filter to reject Ident packets rather than just dropping them. Another IPFilter rule will handle this:

```
iptables -I INPUT -p tcp --➥
dport ident -j REJECT ➥
--reject-with tcp-reset
```

This rule tells the requesting computer that the requested service is not available. Instead of waiting for the request to timeout, the remote machine can then quit establishing the connection without additional delay. ■

INFO

[1] Marc André Selig, "Herding Daemons: Server Management with inetd and Xinetd": Linux Magazine #52.

[2] RFC 1413, Ident protocol: *http://www.faqs.org/rfcs/rfc1413.html*