Part 2: Setting up and Using the Finger Server

Finger Pointing

The finger service can be used to let other people know about your activities and appointments, without having to launch a complex groupware application. The protocol is amazingly simply – and that is one reason why we will be using finger as our entrée into the world of Server administration.

BY MARC ANDRÉ SELIG

Flexible working hours can be a pain sometimes. You end up talking to an answer phone while a colleague is still enjoying breakfast, or has already gone home. Some people simply pin their schedules to their office doors, which is quite useful, unless your office happens to be in a completely different part of your office building. There is a lot to be said for groupware products if you need to organize a meeting and co-ordinate schedules for a group of colleagues.

Although this might surprise some people, Outlook and co. were not the first groupware products on the scene – in fact, the finger protocol was originally specified way back in 1977 [1]. Of course, modern groupware applications offer a lot more, but in many cases finger is quite adequate. In addition, it tells you a lot about the typical Unix philosophy.

How Finger Works

Finger's basic design is wonderfully simple, you might even say primitive – and thus, a perfect example of a client/server service. Users on a system store information about themselves in simple text files. A daemon processes the requests and supplies information on those users.

If you want to use this service to publish information about yourself, you create a file called *.plan* or *.project* in your home directory. The main difference is that a *.project* file traditionally consists of a single line, whereas a *.plan* file provides you with more scope. The *.plan* file is where you would store calendar data or a PGP public key.

Security and the Finger Service

Chapter 3 of RFC 1288 provides an usual amount of detail on security aspects, especially considering the fact that the document is over ten years old (December 1991). Some of the more interesting sections are still applicable to more modern forms of networks, such as dynamic websites or web services.

When they receive a request, some finger servers call a user configurable external program. The RFC says: "Implementing this feature may be more trouble than it is worth, since there are always bugs in operating systems, which could be exploited via this type of mechanism."

Of course many developers and admins simply ignore this warning today, keeping their Apaches with Mod_perl and Mod_php, and solemnly swearing to apply the appropriate security patches, when they get round to it.

The finger daemon adds system data to the information posted by a user, such as the shell, the last login time, or even information on unread mail. Listing 1 shows the results of a sample query *finger Selig@localhost* on my own host.

The daemon parses */etc/passwd* for the username. This is where it retrieves the full name, and the user's room and phone numbers. These three snippets of information are stored in the so-called GECOS field. GECOS is the acronym for General Electric Comprehensive Operating System, which was a fairly widespread operating system around 1970. Although GECOS is insignificant in retrospect, this field with its "human" data migrated from GECOS to Unix and retains the name to indicate its parentage. If you want to change this information, you should call *chfn* (change finger), rather than editing the */etc/passwd* file directly.

Stop finger pointing

Finger allows end-users either to encourage other users to point (the finger daemon) at them, or to prevent finger access. You can create a file called *.nofinger* in your home directory to reject the BSD daemon typical to so many Linux systems without any qualms. You can also prevent finger activity more or less

Listing 1: Finger request

```
01 Login: mas      Name: Marc
   Andre Selig
02 Directory: /home/mas    Shell:
   /bin/bash
03 On since Mon Sep  8 11:20
   (CEST) on pts/0 from :0
04    50 seconds idle
05 On since Mon Sep  8 16:49
   (CEST) on pts/1 from :0
06 No mail.
07 Project:
08 I'm busy with an article for
   Linux Magazine at present.
09 Plan:
10 This
11 is
12 my
13 .plan
14 it can contain more than just
   one line.
```

accidentally by assigning incorrect file permissions. *.plan* and *.project* need to be globally readable, and the daemon expects at least look-up privileges for your home directory (*chmod 711 ~* and *chmod 644 ~/.plan*).

The Daemon

The finger server is included with most distributions, but not installed by default. In this case, you will need to add the *finger* and *finger-server* packages. If you are having trouble finding the package, you can search for *bsd-finger* to locate the sourcecode.

Installing the server will not tell it to launch automatically. Linux distinguishes between two different categories of server. One of them runs continuously as a process, can react immediately to incoming connections and assign resources independently as required. But finger belongs to the other category. When required, the daemon is launched by a central script called *inetd* (the so-called Internet super-server) or *xinetd*, allowing it to respond to current requests.

The latter method has its advantages. Servers are more simple to program, more stable (as an instance is run for each new request), and do not use any resources while inactive.

You can check the */etc/inetd.conf* or */etc/xinetd.d/** files to find out whether your distribution uses the traditional *inetd* or the more modern *xinetd* approach. The finger entry in */etc/inetd.conf* (see Listing 2) is often preconfigured, but disabled by the hash sign #, at the start of the line. You can simply delete the hash sign to enable the server.

Listing 2: The finger daemon in inetd

```
finger  stream  tcp  nowait  nobody  /usr/sbin/in.fingerd  in.fingerd
```

inetd.conf reads like a table where each line represents a specific service that the Internet super-server enables. The individual columns provide more detail about the service.

Protocol Issues

The first column in the line contains the name of the service. It is important to inetd that this name uniquely identifies the IP port. The operating system looks in */etc/services* to discover the port assigned to the symbolic name.

The second column contains the socket type: *stream* indicates a connection-oriented service; *dgram* (datagram) a connectionless service. Connection oriented services use a datastream to exchange data and expect the protocol to provide a reliable transport service. Connectionless services simply transmit individual packets that may go astray or arrive in the wrong order.

To illustrate the difference you might like to compare a phone call with a postcard. In the case of a phone call, the caller expects the conversation to be transmitted without any omissions or errors. In contrast to this, when you mail a postcard, you simply drop it in the mailbox and hope that it will arrive some time. And you are not really surprised if it does go astray.

The third column in */etc/inetd.conf* describes the protocol. This column typically contains either *tcp* for TCP/IP (connection oriented) and *udp* for UDP/IP (connectionless). This would seem to make the column redundant at first glance – but this is not true. Remember that TCP/IP is not the only protocol family. Just like the service name, the entry in the third column is again a symbol that Linux will translate into a number, by referring to the */etc/protocols* file.

To Wait or Not to Wait?

The fourth column tells *inetd* to wait for the current instance of the server program to terminate (*wait*), or not to wait (*nowait*). In the latter case, inetd immediately launches a new server process, when it receives a request for the service.

This may seem superfluous at first glance, but it is an important decision. The entry for the finger service is *nowait*. In other words, when inetd receives a request, it launches a finger server. The server returns a response and terminates. If inetd receives a second request, while the first request is being processed, it immediately calls a second instance of the finger server, which then handles the request.

This would be different for a connectionless service. In this case, inetd cannot know whether an incoming packet belongs to the original request, or if it should launch a second instance of the server. Connectionless services typically expect a *wait* entry in this column.

The fifth column of *inetd.conf* contains the name of the Unix user account that inetd will run the service as. The super-daemon does not automatically launch each server as root; instead you can specify the privileges assigned to the server program.

Column six contains the path to the server executable. Any following fields contain command line arguments for the server, where the first argument (argument number zero) repeats the name of the program.

Too much of a good thing?

Even if you get the configuration right, and everything works perfectly, finger can still provide a potential attacker with a lot of information about your network, the computers on the network, their function, and the social networks within your enterprise. Who works when? On what? And with whom? Which computers do not have any user accounts?

Servers are typically well secured. A machine with a whole bunch of user accounts is just looking to be attacked. Machines with masses of shells often reveal unexpected security holes, although they may be allowed quite generous access to a company network.

Finger also tells you when system administrators are asleep or on vacation, and thus unable to respond to attacks. Finger may even show you the fill or stock levels of network attached coffee machines, or coke and confectionery vending machines. This in turn tells you a lot about the staff's attitude to work.

All of this information can be obtained by other means, but it would mean the attacker putting more effort into footprinting. This is the official reason for most sites doing without finger today. However, the universal use of web servers certainly plays an important role in this area – often with far more serious consequences than a straightforward tool like finger.

If your distribution uses xinetd, it should have a minimal configuration file for finger */etc/xinetd.d/finger* (see Listing 3). There are no major differences in the actual content, although inexperienced users will probably prefer the enhanced readability of this format.

Any changes made to */etc/inetd.conf* or */etc/xinetd.d* typically apply after re-starting the superserver. You can *kill -HUP processID* to parse and apply its new configuration at runtime.

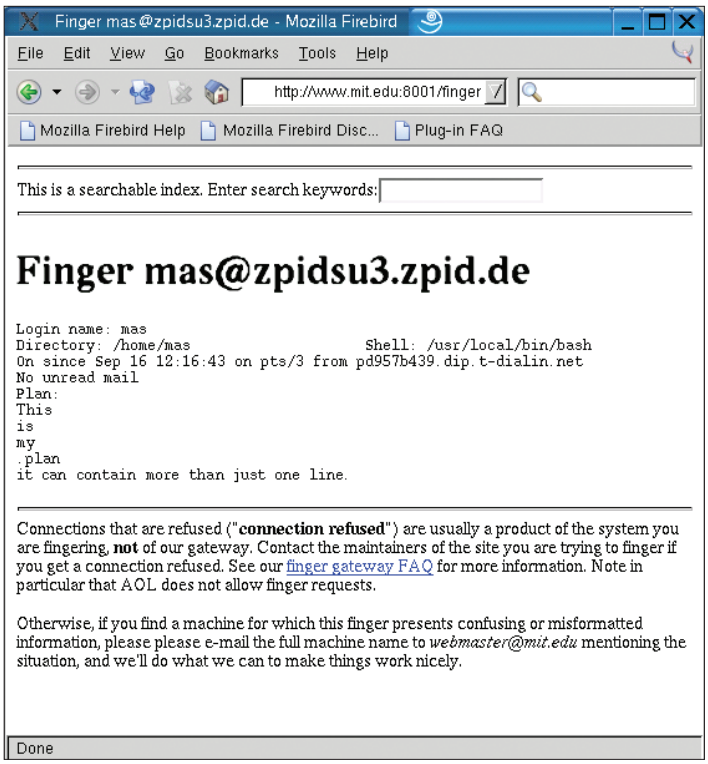

Figure 1: If you want to add a finger request to your homepage, you can use a Web interface to do so. As the protocol is extremely simple, it does not take long to develop a front-end

The Finger Protocol

You do not need a protocol analyzer to watch finger at work on a network, in fact, you do not even need a *finger* client. Instead, you can simply direct finger to the server using a simple Telnet client. If you prefer to read the specification, check out RFC 1288 [2] for the current version.

The server runs on port 79, just one below the HTTP port, 80. It uses a connection oriented protocol, TCP/IP, for transmissions, but not UDP/IP, which is used by connectionless services, such as DNS, NTP, or Syslog. To use finger, you will need to lift any firewall or packet filter restrictions for port 79. You can easily check the availability of the server by typing *netstat -tan | grep :79* – this should produce at least one line of output indicating that the TCP service on port 79 is in the *LISTEN* state.

Type the following command to talk to the daemon:

Listing 3: The finger daemon in xinetd

```
service finger
{
 socket_type     = stream
 wait = no
 user = nobody
 server = /usr/sbin/in.fingerd
 disable = no
}
```

```
telnet localhost 79
```

The Telnet client will then output three lines of data:

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

Telnet has opened up a connection to 127.0.0.1 (that is localhost). The Escape character is for the Telnet client. The daemon is now waiting for individual instructions. Try typing *mas*. The server will interpret this search key both as a Linux username and as a full name. Any matches for these variants are returned across the open TCP connection (see Listing 1), which the daemon then closes.

Is anyone there?

If you simply press [Return], rather than typing an entry in the command line, the finger daemon will return a list of logged on users. Recursive requests are also permissible. A recursive request will tell the finger server to connect to another finger server, and retrieve the requested data from that server. This is useful for firewalls. This type of request uses the *User@Hostname* format. The technique even works across multiple finger servers. You can use an @ sign to separate any additional hostnames in the request.

Each finger command line ends with CRLF (Carriage Return plus Line Feed), which is typical of a TCP connection. Although Unix and Linux use a simple Line Feed to separate newlines, nearly every TCP based protocol uses CRLF. The idea is to allow systems with different architectures to talk to each other, such as Windows computer, which use CRLF natively to keep things simple, or Apple Macintosh machines that only use CR as a newline character.

The end-of-line character is referred to by various names in manpages and many how-tos. As you can type a Line Feed by pressing [Control] + J by default, you may see references to *^J*. C and related languages use *\n*. Newline is shown hex 0x0a, or decimal 10, in ASCII code. Carriage Return is referred to variously as *^M* and *\r*, hex 0x0d, or decimal 13.

Finger without Fear

Before you actually run a finger server on your network, make sure that you read the "Security and the Finger Service" insert. If you are familiar with finger, you will know that you really have nothing to fear – finger has a lot of good points. ■

INFO

[1] RFC 742, the original protocol specification from 1977: *http://www.ietf.org/rfc/rfc742.txt*

[2] RFC 1288, the current version of the finger protocol specs: *http://www.ietf.org/rfc/rfc1288.txt*

[3] Linux manpages as additional reference material for this article: finger(1), in.fingerd(8), chfn(1) and ascii(7).