

# Configuring and Using an FTP Proxy

By Mick Bauer

(Reprinted from the Linux Journal)

Running a public FTP site securely can be difficult. Taking full advantage of the security features supported by your FTP server application of choice can be a chore, and even then there's a good chance that sooner or later vulnerabilities will come to light making all that work for naught. So what else can you do?

One important technique is to run an FTP proxy on your firewall. Whereas the standard Netfilter code in the Linux kernel only inspects packets, an FTP proxy lets your firewall act as an intermediary in all FTP transactions. This increases your protection against buffer overflows and many other kinds of FTP attacks. It also allows you to restrict which FTP commands are executed by FTP clients.

This month I explain how to run SuSE's free (and non-SuSE-Linux-specific) Proxy-Suite FTP proxy on your Linux firewall, adding transparent but strong protection to all your FTP transactions.

## Getting and Installing proxy-suite

If you run SuSE Linux, you can install the package proxy-suite, which installs a binary copy of ftp-proxy along with its configuration file and startup script. If you wish to use ftp-proxy as a transparent proxy, or if you want ftp-proxy to perform LDAP authentication, you'll need the latest version (1.9 as of this writing).

To run the latest version or use ftp-proxy on non-SuSE distributions, your best bet is to compile it yourself from source code, available at [ftp.suse.com/pub/projects/proxy-suite/src](http://ftp.suse.com/pub/projects/proxy-suite/src) [1].

Building from Source

Complete instructions on building and installing ftp-proxy are provided in the file INSTALL. By default, the configure script will check for libwrap, libldap and whether your system supports regular expressions. On my Red Hat 7.3 system, libwrap was present but caused a compile-time error, so I disabled libwrap like this:

```
# ./configure --without-libwrap
```

and ftp-proxy compiled properly. However, this wasn't necessary when I compiled ftp-proxy on my SuSE 7.1 system (obviously, SuSE's and Red Hat's libwrap packages differ).

After building ftp-proxy and installing it and its documentation, you'll probably want a startup script for your new proxy. Included with ftp-proxy's source (in the directory ftp-proxy/) is a sample script, rc.script, which is explained in the accompanying file rc.script.txt.

On SuSE systems, you simply can copy rc.script to /etc/init.d and optionally create a symbolic link to it from /usr/sbin. Rename the script /etc/init.d/ftp-proxy, and name the symbolic link /usr/sbin/rcftp-proxy. If you run SuSE 7.x, you'll also need to add this line to /etc/rc.config:

```
START_FTP_PROXY="yes"
```

For non-SuSE distributions, the example rc.script will need to be heavily tweaked, because much of it is SuSE-specific. Look at other scripts in your distribution's init.d directory for examples. Once you've figured out how, I strongly encourage you to send your hacked script to Marius Tomaschewski

Revised December 1, 2002

Page 1 of 6

# Configuring and Using an FTP Proxy

By Mick Bauer

(Reprinted from the Linux Journal)

(mt@suse.de [2]), one of the major contributors to FTP-Proxy, so others may benefit from your brilliance.

## Configuring ftp-proxy

Once you've installed ftp-proxy from source or from a SuSE package, it's time to configure it. Most configurable parameters are kept in /etc/proxy-suite/ftp-proxy.conf (or, if you installed from source, in /usr/local/etc/proxy-suite/ftp-proxy.conf). Before diving into ftp-proxy.conf, however, you've got a couple of odds and ends to attend to.

First, you need a new, unprivileged user account for the proxy daemon to use. On my system I created such a user, ftpproxy, like this:

```
bash-# useradd -u 65500 -g nogroup -d /var/ftp-proxy/rundir -s /bin/false ftpproxy
```

No one should log in as this user, so be sure also to put an asterisk in the password field of the proxy user's line in /etc/shadow:

```
ftpproxy:*:12345:0:99999:7:0:::
```

Next, you'll need to build a chroot jail in which ftp-proxy's child processes can work. For SuSE users this is easy; ftp-proxy's startup script will do this for you if invoked with the chroot command:

```
bash-# /etc/init.d/ftp-proxy chroot
```

Even if you don't run SuSE, it's fairly simple to reverse engineer the example script (the rc.script mentioned earlier) to figure out how to do this. The long and short of it is that the customary ftp-proxy chroot jail is /var/ftp-proxy/rundir, and it should contain copies of the libraries and files ftp-proxy uses, plus its own dev/log special file to which your local syslog daemon can listen.

To point your syslog daemon to the chrooted log device, simply add an -a parameter to its startup script so that syslog is started:

```
syslog -a /var/ftp-proxy/rundir/dev/log
```

On SuSE systems the customary way to do this is in /etc/rc.config via the SYSLOGD\_PARAMS variable. You can specify multiple -a statements if, for example, you're also receiving logs from a chrooted named.

## ftp-proxy.conf

And now, finally, it's time to configure your proxy daemon. As I mentioned, this is done in the file ftp-proxy.conf, which resides either in /etc/proxy-suite or in /usr/local/etc/proxy-suite. You may be confused or annoyed by SuSE's use of the term "suite" to refer to a single application. Hopefully, additional proxies will be completed soon, and if they're as useful as ftp-proxy, I, for one, will forgive them for this minor conceit.

The quickest way to explain this file is to list a brief example and dissect it (see Listing 1).

# Configuring and Using an FTP Proxy

By Mick Bauer

(Reprinted from the Linux Journal)

## Listing 1. ftp-proxy.conf

```
ServerType      standalone
User           ftpproxy
Group          nogroup
LogDestination daemon
# LogLevel     INF
# DBG is useful for
# troubleshooting
PidFile        /var/run/ftp-proxy.pid
# ServerRoot   /var/ftp-proxy/rundir
AllowMagicUser no
AllowTransProxy yes
DestinationAddress 192.168.1.2
ValidCommands  USER, PASS, PWD, CWD, CDUP, PORT, PASV, RETR, TYPE, REST, ABOR,
LIST, NLST, STAT, QUIT
```

The first parameter, `ServerType`, determines whether to run `ftp-proxy` as a standalone daemon or from `inetd`. Although I've been calling it a daemon, `ftp-proxy` can be run either way. I personally avoid running `inetd` or even `xinetd` on my public servers, because that way I don't need to disable the unnecessary things that tend to get run by default, and because of the performance benefit of running things as daemons. If your needs are different, you can set `ServerType` to `inetd` (which also works if you run `xinetd` rather than `inetd`).

`User` and `Group`, obviously enough, determine the UID and GID under which `ftp-proxy` runs after initialization. It's a good idea to set these to an unprivileged UID and GID in order to lessen the consequences of an attacker somehow hijacking an `ftp-proxy` process.

`LogDestination` specifies where `ftp-proxy` should send log messages. This can be either daemon (the local syslog facility), a file or a pipe. `LogLevel` determines the quantity of information to be logged; for most users the default of `INF` is best, but `DBG` (the maximum setting) is useful for troubleshooting.

`PidFile` tells `ftp-proxy` where to store the process ID of its master process. This is used by the startup script when it's invoked with the stop command and upon system halt. It isn't used, however, if `ftp-proxy` is run in `inetd` mode.

`ServerRoot` specifies the path to `ftp-proxy`'s chroot jail. Leave it commented out if you don't want to run `ftp-proxy` chrooted (see the "Problem with 1.9 and chroot" Sidebar).

## Transparent Proxying

The next three commands in Listing 1 are important. They determine whether your proxy will be transparent. In most situations, a transparent proxy is preferable. End users won't need to configure their FTP client software to explicitly support the proxy. To achieve this, `ftp-proxy` works in conjunction with the kernel's Netfilter code, which redirects FTP packets to your proxy daemon rather than sending them to the host to which they're actually addressed.

# Configuring and Using an FTP Proxy

By Mick Bauer

(Reprinted from the Linux Journal)

When ftp-proxy receives FTP client packets that have been redirected in this way, it uses their destination IP as the destination of the new FTP connection it initiates to the desired FTP server. The parameter DestinationAddress specifies the default destination to use.

If you want to allow users to use the proxy non-transparently, i.e., by initiating their FTP sessions directly to the proxy, set the parameter AllowMagicUser to “yes”, but I do *not* recommend doing so if your proxy is to be used by external users, as in the case of a public FTP. AllowMagicUser will cause your proxy to act as an open proxy that external users may use to connect to *other, external* FTP servers, possibly for the purpose of attacking them.

If you've configured Netfilter to accept connections to the proxy from trusted (internal) users only, however, and you set AllowMagicUser to “yes”, users will be able to specify their FTP destination by attaching it to their user name with an @ sign, e.g., mick@ftp.wiremonkeys.org. AllowMagicUser may be used regardless of whether AllowTransProxy is set to yes or no. But note that if it's set to no and AllowMagicUser is too, *all* FTP sessions will use DestinationAddress.

Other parameters include MaxClientsString and DestinationTransferMode. See the ftp-proxy.conf(8) man page for the complete list and for more information on the ones we've covered here.

## Configuring Netfilter for Transparent Proxying

For transparent proxying to work you need to use iptables to redirect FTP packets to the local proxy (i.e., you need to run Netfilter on your proxy host, which this article assumes you're doing), and of course, you'll need rules allowing FTP connections to and from the proxy. You will *not*, however, need any rules in the FORWARD chain.

First, you'll need to load several modules for your Linux 2.4 firewall to support transparent proxying: ipt\_conntrack\_ftp and ip\_nat\_ftp are required for FTP connection tracking; ipt\_REDIRECT is required for the REDIRECT rule target. Most distributions' stock 2.4 kernels include these modules.

Once the modules are loaded, you can add firewall rules like these to your Netfilter startup script (Listing 2).

### Listing 2. iptables: Commands for Transparent FTP Proxying

```
iptables -t nat -A PREROUTING -p tcp -i eth2 --dport 21 -j REDIRECT
iptables -t nat -A PREROUTING -p tcp -i eth0 --dport 21 -j REDIRECT

# snip...

iptables -A INPUT -p tcp -d $PUBLIC_FTP --dport 21 -m state --state NEW,RELATED -j
ACCEPT
iptables -A INPUT -p tcp -s $INTERNAL_HOSTS --dport 21 -m state --state
NEW,RELATED -j ACCEPT

# snip...
```

# Configuring and Using an FTP Proxy

By Mick Bauer

(Reprinted from the Linux Journal)

```
iptables -A OUTPUT -p tcp -d $PUBLIC_FTP --dport 21 -m state --state NEW,RELATED -j ACCEPT
iptables -A OUTPUT -p tcp -o eth2 --dport 21 -m state --state NEW,RELATED -j ACCEPT
```

The first two commands of Listing 2, instruct the firewall to redirect all packets received on its external and internal interfaces (eth2 and eth0, respectively) that have a destination port of TCP 21 (the FTP server port). Note that these packets won't be rewritten (mangled) in any way; they'll simply be redirected to the local FTP proxy daemon.

The third and fourth commands in Listing 2 tell the firewall to accept all incoming packets sent to TCP port 21 of the public FTP server (where the variable PUBLIC\_FTP contains its IP address) and all incoming FTP packets sent by internal users (where the variable INTERNAL\_HOSTS contains an IP range in CIDR notation, e.g., 192.168.99.0/24). Per the first two lines, any packets matching lines three and four will be diverted to the local proxy.

The fifth and sixth lines in Listing 2 allow the local ftp-proxy daemon to initiate proxied FTP connections to the specified public FTP server and to external FTP servers (i.e., hosts reachable from its external Ethernet interface, in this example, eth2).

The lines in Listing 2 do *not* form a self-contained Netfilter rulebase. They represent the lines you could add to an existing script already properly configured for NAT, etc., and already containing definitions for the variables PUBLIC\_FTP and INTERNAL\_HOSTS. It's good practice to use custom variables like this to make your rules more readable.

## Restricting FTP Commands

Now we return to ftp-proxy.conf (Listing 1) and one of ftp-proxy's most important features: ValidCommands. This is a comma-delimited list of FTP commands the proxy will allow. The list may span multiple lines if you end each line (except for the last) with a backslash (\). In the ValidCommands statement at the bottom of Listing 1, ftp-proxy has been configured to allow FTP directory navigation commands (PWD, CWD, CDUP) and FTP read commands (LIST, NLST, RETR), plus some additional administrative commands such as MODE, PORT and PASV.

Space does not permit me to explain all of these in depth, other than to say that these aren't end-user FTP client commands; they're FTP protocol commands as specified in RFC 959 (see ftp.isi.edu/in-notes/rfc959.txt [7]). These are the commands that FTP client and server applications use with each other. See Table 1 for a summary.

**Table 1. FTP Commands Specified by RFC 959**

RFC	959	
Command		Description
USER		Used to specify user name.
PASS		Used to specify password.
PWD		Print working directory.

# Configuring and Using an FTP Proxy

By Mick Bauer

(Reprinted from the Linux Journal)

CWD	Change working directory.
CDUP	Change to working directory's parent directory.
TYPE	Set data type: IMAGE (binary), ASCII, EBCDIC or L (local
MODE	Set data-flow type to stream, block or compressed;
PASV	Tells server to prepare to receive a data channel in "passive"
PORT	Tells server to open (send) a data channel in "active"
RETR	Retrieve (read) a file.
STOR	Send (write) a file.
APPE	Send a file; if it already exists then append it to the extant
REST	Fast-forward to specified file position. Must be followed by a
RNFR	Rename from: must be followed on the same line by a RNTO
RNTO	Rename to: must be preceded on the same line by a RNFR
	command.
STOU	Send (write) a file; change its filename if it already exists.
ABOR	Cancel previous command and any resulting activity
DELE	Delete file.
RMD	Remove directory.
MKD	Make directory.
LIST	List filenames, with file info.
NLST	List filenames only.
SYST	Ask server its system type.
STAT	List attributes of a specified file or the print status of
QUIT	Terminate session.

One limitation of ftp-proxy is that it isn't possible to set different command restrictions for external users than for internal users. Be careful, therefore, with ValidCommands. If your internal users need to send files to FTP servers, you won't be able to restrict the STOR or STOU commands (i.e., you'll need to include them in ValidCommands), which means you'll need to make sure your read-only public FTP server is itself configured to disregard them.

That isn't such a bad thing. Regardless of how ftp-proxy is configured, you still need to configure your FTP servers to protect *themselves* as much as possible.

## Conclusion

An FTP proxy adds an important layer of security between the bad guys and your public FTP servers. I've shown you the basics of setting up a transparent FTP proxy using SuSE's proxy-suite, but it supports many other worthwhile features we haven't covered here. See the Resources section for pointers to additional information. Good luck!