Transparent Proxy with Linux and Squid mini-HOWTO

Daniel Kiracofe

v1.15, August 2002

This document provides information on how to setup a transparent caching HTTP proxy server using only Linux and squid.

---

1. Introduction

1.1 Comments

Comments and general feedback on this mini HOWTO are welcome and can be directed to its author, Daniel Kiracofe, at drk@unxsoft.com.

1.2 Copyrights and Trademarks

1.3 #include <disclaimer.h>

No warranty, expressed or implied, etc, etc, etc...

---

2. Overview of Transparent Proxying

2.1 Motivation

In ``ordinary'' proxying, the client specifies the hostname and port number of a proxy in his web browsing software. The browser then makes requests to the proxy, and the proxy forwards them to the origin servers. This is all fine and good, but sometimes one of several situations arise. Either

- You want to force clients on your network to use the proxy, whether they want to or not.
- You want clients to use a proxy, but don't want them to know they're being proxied.
- You want clients to be proxied, but don't want to go to all the work of updating the settings in hundreds or thousands of web browsers.

This is where transparent proxying comes in. A web request can be intercepted by the proxy, transparently. That is, as far as the client software knows, it is talking to the origin server itself, when it is really talking to the proxy server. (Note that the transparency only applies to the client; the server knows that a proxy is involved, and will see the IP address of the proxy, not the IP address of the user. Although, squid may pass an X-Forwarded-For header, so that the server can determine the original user's IP address if it groks that header).

Cisco routers support transparent proxying. So do many switches. But, (surprisingly enough) Linux can act as a router, and can perform transparent proxying by redirecting TCP connections to local ports. However, we also need to make our web proxy aware of the affect of the redirection, so that it can make connections to the proper origin servers. There are two general ways this works:

The first is when your web proxy is not transparent proxy aware. You can use a nifty little daemon called transproxy that sits in front of your web proxy and takes care of all the messy details for you. transproxy was written by John Saunders, and is available from

ftp://ftp.nlc.net.au/pub/linux/www/ or your local metalab mirror. transproxy will not be discussed further in this document.

A cleaner solution is to get a web proxy that is aware of transparent proxying itself. The one we are going to focus on here is squid. Squid is an Open Source caching proxy server for Unix systems. It is available from www.squid-cache.org

Alternatively, instead of redirecting the connections to local ports, we could redirect the connections to remote ports. This is discussed in the Transparent Proxy to a Remote Box section. Readers interested in this approach should skip down to that section. Readers interested on doing everything on one box can safely ignore that section.

2.2 Scope of this document

This document will focus on squid version 2.4 and Linux kernel version 2.4, the most current stable releases as of this writing (August 2002). It should also work with most of the later 2.3 kernels. If you need information about earlier releases of squid or Linux, you can find some earlier documents at http://users.gurulink.com/transproxy/. Note that this site has moved from it's previous location.

If you are using a development kernel or a development version of squid, you are on your own. This document may help you, but YMMV.

Note that this document focuses only on HTTP proxing. I get many emails asking about transparent FTP proxying. Squid can't do it. Now, allegedly a program called Frox can. I have not tried this myself, so I cannot say how well it works. You can find it at http://www.hollo32.fsnet.co.uk/frox/.

I only focus on squid here, but Apache can also function as a caching proxy server. (If you are not sure which to use, I recommend squid, since it was built from the ground up to be a caching proxy server, Apache's caching proxy features are more of afterthought additions to an already existing system.) If you want use Apache instead of squid: follow all the instructions in this document that pertain to the kernel and iptables rules. Ignore the squid specific sections, and instead look at http://lupo.campus.uniroma2.it/progetti/mod_tproxy/ for source code and instructions for a transparent proxy module for Apache (thanks to Cristiano Paris (c.paris@libero.it) for contributing this).

2.3 HTTPS

Finally, as far as transparently proxing HTTPS (e.g. secure web pages using SSL, TSL, etc.), you can't do it. Don't even ask. For the explanation, do a search for 'man-in-the-middle attack'. Note that you probably don't really need to transparently proxy HTTPS anyway, since squid can not cache secure pages.

2.4 Proxy Authentication

You cannot use Proxy Authentication transparently. See the Squid FAQ for (slightly) more details.

---

3. Configuring the Kernel

First, we need to make sure all the proper options are set in your kernel. If you are using a stock kernel from your distribution, transparent proxying may or may not be enabled. If you are unsure, the best way to tell is to simply skip this section, and if the commands in the next section give you weird errors, it's probably because the kernel wasn't configured properly.

If your kernel is not configured for transparent proxying, you will need to recompile. Recompiling a kernel is a complex process (at least at first), and it is beyond the scope of this document. If you need help compiling a kernel, please see The Kernel HOWTO

The options you need to set in your configuration are as follows (Note: if you prefer modules, some (but not all) of these can be built as modules. Luckily, everything that is not modularizable is probably got in your kernel anyway.)

- Under General Setup
  - Networking support
  - Sysctl support
- Under Networking Options
  - Network packet filtering
  - TCP/IP networking
- Under Networking Options -> IP: Netfilter Configuration
  - Connection tracking
  - IP tables support
  - Full NAT
  - REDIRECT target support
- Under File Systems
  - /proc filesystem support

You must say NO to ``Fast switching" under Networking Options.

Once you have your new kernel up and running, you may need to enable IP forwarding. IP forwarding allows your computer to act as a router. Since this is not what the average user wants to do, it is off by default and must be explicitly enabled at run-time. However, your distribution might do this for you already. To check, do ``cat /proc/sys/net/ipv4/ip_forward". If you see ``1" you're good. Otherwise, do ``echo '1' > /proc/sys/net/ipv4/ip_forward". You will then want to add that command to your appropriate bootup scripts (depending on your distribution, these may live in /etc/rc.d, /etc/init.d, or maybe somewhere else entirely).

---

4. Setting up squid

Now, we need to get squid up and running. Download the latest source tarball from www.squid-cache.org. Make sure you get a STABLE version, not a DEVEL version. The latest as of this writing was squid-2.4.STABLE4.tar.gz. Note that AFAIK, you must have squid-2.4 for linux kernel 2.4. The reason is that the mechanism by which the process determines the original destination address has changed from linux 2.2, and only squid-2.4 has this new code in it. (For those of you who are interested, previously the getsockname() call was hacked to provide the original destination address, but now the call is getsockopt() with a level of SOL_IP and an option of SO_ORIGINAL_DST).

Now, untar and gunzip the archive (use ``tar -xzf <filename>''). Run the autoconfiguration script and tell it to include netfilter code (``./configure --enable-linux-netfilter''), compile (``make'') and then install (``make install'').

Now, we need to edit the default squid.conf file (installed to /usr/local/squid/etc/squid.conf, unless you changed the defaults). The squid.conf file is heavily commented. In fact, some of the best documentation available for squid is in the squid.conf file. After you get it all up and running, you should go back and reread the whole thing. But for now, let's just get the minimum required. Find the following directives, uncomment them, and change them to the appropriate values:

- httpd_accel_host virtual
- httpd_accel_port 80
- httpd_accel_with_proxy on
- httpd_accel_uses_host_header on

Next, look at the cache_effective_user and cache_effective_group directives. Unless the default nobody/nogroup has been created on your system (AFAIK, it is not created out of the box on many popular distributions, including RH7.1), you'll either need to create those, or create another username/group for squid to run under. I strongly recommend that you create a username/group of squid/squid and run under that, but you could use any existing user/group if you want.

Finally, look at the http_access directive. The default is usually ``http_access deny all''. This will prevent anyone from accessing squid. For now, you can change this to ``http_access allow all'', but once it is working, you will probably want to read the directions on ACLs (Access Control Lists), and setup the cache such that only people on your local network (or whatever) can access the cache. This may seem silly, but you should put some kind of restrictions on access to your cache. People behind filtering firewalls (such as porn filters, or filters in nations where speech is not very free) often ``hijack'' onto wide open proxies and eat up your bandwidth.

Initialize the cache directories with ``squid -z'' (if this is a not a new installation of squid, you should skip this step).

Now, run squid using the RunCache script in the /usr/local/squid/bin/ directory. If it works, you should be able to set your web browser's proxy settings to the IP of the box and port 3128 (unless you changed the default port number) and access squid as a normal proxy.

For additional help configuring squid, see the squid FAQ at

---

5. Setting up iptables (Netfilter)

iptables is a new thing for Linux kernel 2.4 that replaces ipchains. If your distribution came with a 2.4 kernel, it probably has iptables already installed. If not, you'll have to download it (and possibly compile it). The homepage is netfilter.samba.org. You make be able to find binary RPMs elsewhere, I haven't looked. For the curious, there is plenty of documentation on the netfilter site.

To set up the rules, you will need to know two things, the interface that the to-be-proxied requests are coming in on (I'll use eth0 as an example) and the port squid is running on (I'll use the default of 3128 as an example).

Now, the magic words for transparent proxying:

- iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 3128

You will want to add the above commands to your appropriate bootup script under /etc/rc.d/. Readers upgrading from 2.2 kernels should note that this is the only command needed. 2.2 kernels required two extra commands in order to prevent forwarding loops. The infastructure of netfilter is much nicer, and only this command is needed.

---

6. Transparent Proxy to a Remote Box

Now, the question naturally arises, if we can do all this nifty stuff redirecting HTTP connections to local ports, could we do the same thing but to a remote box (e.g., the machine with squid running is not the same machine as iptables is running on). The answer is yes, but it takes a little different magic words. If you only want to redirect to the local box (the normal case), skip this section.

For the purposes of example commands, let's assume we have two boxes called squid-box and iptables-box, and that they are on the network local-network. In the commands below, replace these strings with the actual IP addresses or name of your machines and network.

I will present two different approaches here.

6.1 First method (simpler, but does not work for some esoteric cases)

First, we need to machine that squid will be running on, squid-box. You do not need iptables or any special kernel options on this machine, just squid. You *will*, however, need the 'http_accel' options as described above. (Previous version of this HOWTO suggested that you did not need those options. That was a mistake. Sorry to have confused people...)

Now, the machine that iptables will be running on, iptables-box You will need to configure the kernel as described in section 3 above, except that you don't need the REDIRECT target support). Now, for the iptables commands. You need three:

- iptables -t nat -A PREROUTING -i eth0 -s ! **squid-box** -p tcp --dport 80 -j DNAT --to **squid-box**:3128
- iptables -t nat -A POSTROUTING -o eth0 -s **local-network** -d **squid-box** -j SNAT --to **iptables-box**
- iptables -A FORWARD -s **local-network** -d **squid-box** -i eth0 -o eth0 -p tcp --dport 3128 -j ACCEPT

The first one sends the packets to squid-box from iptables-box. The second makes sure that the reply gets sent back through iptables-box, instead of directly to the client (this is very important!). The last one makes sure the iptables-box will forward the appropriate packets to squid-box. It may not be needed. YMMV. Note that we specified '-i eth0' and then '-o eth0', which stands for input interface eth0 and output interface eth0. If your packets are entering and leaving on different interfaces, you will need to adjust the commands accordingly.

Add these commands to your appropriate startup scripts under /etc/rc.d/

(Thanks to Giles Coochey for help writing this section).

6.2 Second method (more complicated, but more general)

Our first shot at this works good, but there is a minor drawback in that HTTP/1.0 connections without the Host header do not get handled properly. Connections that are fully or partially HTTP/1.1 compliant work fine. As most modern web browsers send the Host header, this is not a problem for most people. However, some small programs or embedded devices may send only very simple HTTP/1.0 requests. If you want to support these, we'll need to do a little more work. Namely, on iptables-box we'll need the following options enabled in the kernel in addition to what was specified above:

- IP: advanced router
- IP: policy routing
- IP: use netfilter MARK value as routing key
- IP: Netfilter Configuration -> Packet mangling
- IP: Netfilter Configuration -> MARK target support

You'll also need the iproute2 tools. Your distribution probably already has them installed, but if not, look at ftp://ftp.inr.ac.ru/ip-routing/

You'll want to use the following set of commands on iptables-box:

- iptables -t mangle -A PREROUTING -j ACCEPT -p tcp --dport 80 -s **squid-box**
- iptables -t mangle -A PREROUTING -j MARK --set-mark 3 -p tcp --dport 80
- ip rule add fwmark 3 table 2

- ip route add default via **squid-box** dev eth1 table 2

Note that the choice of firewall mark (3) and routing table (2) was fairly arbitrary. If you are already using policy routing or firewall marking for some other purpose, make sure you choose unique numbers here. Otherwise, don't worry about it.

Next, squid-box. Use this command, which should look remarkably similar to a command we've seen previously.

- iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 3128

As before, add all of these commands to the appropriate startup scripts.

Here is a brief explanation of how this works: in method one, we used Network Address Translation to get the packets to the other box. The result of this is that the packet gets altered. This alteration is what causes some kinds of clients mentioned above to fail. In method two, we use a magic thing called policy routing. The first thing we do is to select the packets we want. Thus, all packets on port 80, except those coming from squid-box itself, are MARKed. Then, when the kernel goes to make a routing decision, the MARKed packets aren't routing using the normal routing table that you access with the ``route'' command but with a special table. This special table has only one entry, a default gateway to squid-box. Thus, the packet is sent merrily on it's way without every having been altered. So, even HTTP/1.0 connections can be handled perfectly. (Thanks to Michal Svoboda for suggesting and helping to write this section)

6.3 Method One: What if iptables-box is on a dynamic IP?

If the iptables-box is on a dynamic IP address (e.g. a dialup PPP connection, or a DHCP assigned IP address from a cable modem, etc.), then you will want to make a slight change to the above commands. Replace the second command with this one:

- iptables -t nat -A POSTROUTING -o eth0 -s **local-network** -d **squid-box** -j MASQUERADE

This change avoids having to specify the IP address of iptables-box in the command. Since it will change often, you'd have to change your commands to reflect it. This will save you a lot of hassle.

---

7. Transparent Proxy With Bridging

Warning, this is really esoteric stuff. If you need it, you'll know. If not, skip this section. Thanks to Lewis Shobbrook (lshobbrook@fasttrack.net.au) for contributing to this section.

If you are trying to setup a transparent proxy on a Linux machine that has been configured as a bridge, you will need to add one additional iptables command to what we had in section 5.

Specifically, you need to explicitly allow connections to the machine on port 3128 (or any other port squid is listening on), otherwise the machine will just forward them over to the other interface like a good little bridge. Here's the magic words:

- iptables -A INPUT -i **interface** -p tcp -d **your_bridge_ip** -s **local-network** --dport 3128 -m state --state NEW,ESTABLISHED -j ACCEPT

Replacing **interface** with the interface that corresponds to **your_bridge_ip** (typically eth0 or eth1). First time bridge users should also note that you'll probably want to repeat the same command with ``3128'' replaced by ``telnet'' if you want to administer your bridge remotely.

---

8. Put it all together

If everything has gone well so far, go to another machine, change it's gateway to the IP of the box with iptables running on it, and surf away. To make sure that requests are really being forwarded through your proxy instead of straight to the origin server, check the log file /usr/local/squid/logs/access.log

---

9. Troubleshooting

There is one problem that occurs often enough to mention here. If you get the following error:

```
/lib/modules/2.4.2-2/kernel/net/ipv4/netfilter/ip_tables.
o init_modules: Device or resource busy Hints: insmod
errors can be caused by incorrect module parameters;
including invalid IO or IRQ parameters.

perhaps iptables or your kernel needs to be upgraded...
```

then you are probably running Red Hat 7.x. The folks at Red Hat, in all their wisdom, decided to load the ipchains module by default on startup. I guess this was for backwards compatibility for those who haven't learned iptables yet. However, the problem is that ipchains and iptables are mutually incompatible. Since ipchains has been secretly loaded by RH, you cannot use iptables commands. To see if this is your problem, do the command ``lsmod'' and look for the module named ``ipchains''. If you see it, that is your problem. The quick fix is to execute the command ``rmmod ipchains'' before you issue any iptables commands. To permanently remove these commands from your startup scripts, the following command should work: ``/sbin/chkconfig --level 2345 ipchains off''. (Thanks to Rasmus Glud for pointing this command out to me).

---

10. Further Resources

Should you still need assistance, you may wish to check the squid FAQ or the squid mailing list at www.squid-cache.org. You may also e-mail me at drk@unxsoft.com, and I'll try to answer your questions if time permits (sometimes it does, but sometimes it doesn't). Please, please, please, send the output of ``iptables -t nat -L'' and relavent portions of any configuration files in your e-mail, or else I will probably not be able to help you out much. And please make sure you've read the whole HOWTO before asking a question. Regrettably, even though this document has been translated to many different languages, I can only answer questions asked in English.