

Squid (WWW Proxy Server)

Jason Healy, Director of Networks and Systems

Last Updated Mar 18, 2008

Contents

1	Squid (WWW Proxy Server)	5
1.1	Introduction	5
1.2	Suffield's Requirements	5
1.3	Hardware	6
1.4	Custom Compilation	7
1.4.1	Preparing The System	7
1.4.2	Kernel Compilation	8
1.4.3	IPTables Compilation	9
1.4.4	Squid Compilation	11
1.5	Configuration	13
1.5.1	Bridge Setup	13
1.5.2	Disk Setup	14
1.5.3	Squid Configuration	15
1.5.4	TPROXY Rules	15
1.5.5	Watching Squid	17
1.5.6	Initialization Scripts	18
1.6	Cache Manager	19
1.6.1	Configuration	19
1.6.2	Security	19
1.7	Squid Statistics	19

1.7.1	Downloading and Installing Calamaris	20
1.7.2	Configuring Calamaris	20
1.7.3	Running Calamaris	21

Chapter 1

Squid (WWW Proxy Server)

Last updated 2008/03/18

1.1 Introduction

A large portion of our internet link is used for web (HTTP) traffic. In addition to popular websites, many key pieces of software (system updates, linux packages, and multimedia programs) make use of HTTP to deliver their content.

To increase the efficiency of our internet connection, we use a **caching web proxy**. A web proxy makes the HTTP request on behalf of the client, and caches the result for later. That way, if multiple users request the same document, the cache can serve its local copy, rather than wasting bandwidth. This has two benefits: users get their content faster (when its cached), and the internet connection isn't bogged down with duplicate requests.

1.2 Suffield's Requirements

At Suffield Academy, we use **Squid** as our proxy server. Briefly, these are the features that Squid supports that make it attractive for us to use:

- Squid can act as an **interception** (or **transparent**) proxy. This means that the clients on our network don't need to be reconfigured; all web

traffic is automatically cached without them needing to do anything. (We put our proxy inline with other network devices, though it is possible to set up routing rules to forward web traffic to the proxy.)

- Squid can use external **redirectors** for every request. This means that we can log or block unwanted sites using 3rd-party blacklists (note that we don't do this currently, as Suffield does not block traffic).

Additionally, redirectors can be used to circumvent requests to unwanted URLs (such as those for advertisements). We use such a program, which makes pages load faster (no need to fetch the ads), and also prevents users from having to see unwanted advertisements.

- Squid has a feature called **delay pools**, which allow us to specify per-user and global transfer rates. This means that we can throttle users who consume more than their fair share of bandwidth. Additionally, we can apply rules to specific types of resources to prevent them from being abused (for example, we can throttle connections that go to <http://www.example.com>).

All of these features make Squid an extremely attractive piece of software. It's relatively straightforward to compile and set up, and it's Free software.

1.3 Hardware

Squid has one basic job: fetch data over the network, and store a copy on the local machine. Information is kept in RAM when possible, and on disk the rest of the time.

Thus, the two most important factors in hardware are the amount of RAM given to Squid, and the speed of the disk subsystem used to store the cache data.

The Squid listserver archives contain numerous messages about the "recommended" machine configuration for Squid. Unfortunately, there is no good answer to this question, as every site's needs are different.

Our first Squid machine was a Pentium III 850MHz with 1.5GB of RAM. It had two 36GB SCSI disks, which we ran in a RAID-1 configuration. The machine performed well, but as our internet connection speed grew, it began to show its age. We especially wanted a faster processor so we could have the flexibility to run filtering software inline with the proxy.

Our current machine is a P4 3.4GHz with 3GB of RAM. The machine has 6 15K SCSI disks. One is used for the system, and the others are used for the Squid cache. As our needs increase, we can add more drives to the system, and split them onto different controller cards. We no longer run the machine in a RAID configuration, as that just brings a speed penalty (the Squid daemon automatically balances load between multiple disks for us).

1.4 Custom Compilation

We use our Squid box as a bridging interception proxy. This means that the machine acts as a bridge, and is inserted "inline" between other network devices (in our case, just before our firewall). Additionally, Squid is configured as an interception proxy, which means that it will grab requests destined for port 80 (the standard HTTP port) and run them through the caching engine. Connections not bound for port 80 are passed straight through the bridge.

This setup is advantageous because it makes the proxy completely invisible to the rest of the network (and the users). If we need to, we can simply take the machine out of the network and everything will continue to work just as before (except that it's not being cached). No settings on the client machines need to change.

However, this setup requires compiling our own Linux kernel, and also our own version of Squid. We'll walk you through the steps below.

Note: it is possible to set up squid to work in a non-bridged mode (all of our custom compilation in this section deals with a bridged version of Squid). This requires some other means of getting the packets to your squid box, such as via WCCP or router next-hop mangling. We used to do this (we had a `route-map` command on our Cisco core switch), but the routing was done in software, and it drove up the load on our core. We've now switched to an inline approach, which doesn't impact performance on our other gear.

Note: these steps assume Debian Linux ("Etch" is the release we're using). If you use a different system you may need to compile according to your distribution's conventions.

1.4.1 Preparing The System

First, set some environment variables so your custom builds will have sane values in the description files:

```
export DEBFULLNAME="Your Name"
export DEBEMAIL="you@example.com"
```

Now we need to download all the components necessary to build packages on our system. Make sure your package repository listing is up-to-date:

```
apt-get update
```

Move into the source directory on the machine:

```
cd /usr/src
```

Now we need to download a special patch for bridging proxy support in the kernel and iptables, and unpack it.

```
wget http://www.balabit.hu/downloads/files/tproxy/obsolete/linux-2.6/cttproxy-2.6.18-2.0.6.tar.gz
tar -zxf cttproxy-2.6.18-2.0.6.tar.gz
```

We are now ready to fetch and build the three other components we need: the kernel, iptables, and squid.

1.4.2 Kernel Compilation

First, fetch the kernel package and all of the packages necessary to build a kernel (you can put the following command all on one line):

```
apt-get install linux-source-2.6.18 kernel-package libncurses5-dev \
fakeroot bzip2 build-essential dpatch devscripts
```

This will put a tarball of the Linux source in /usr/src. Go there and unpack it:

```
cd /usr/src
tar -jxf linux-source-2.6.18.tar.bz2
```

Now move into the Linux source directory:

```
cd linux-source-2.6.18
```

You need to apply the patches included with the tproxy download you got earlier:

```
for patch in ../cttproxy-2.6.18-2.0.6/patch_tree/*.patch
do patch -p1 < $patch
done
```

Now it's time for a standard Debian kernel build. I suggest making based on the existing config:

```
make oldconfig
```

(You may also use `make menuconfig` for a graphical interface.)

That will ask you about the new features added by the patch. You want to enable the following:

```
Transparent proxying (IP_NF_TPROXY)
tproxy match support (IP_NF_MATCH_TPROXY)
TPROXY target support (IP_NF_TARGET_TPROXY)
```

```
NAT reservations support (IP_NF_NAT_NRES)
```

You're now ready to build the kernel:

```
make-kpkg clean
make-kpkg --rootcmd fakeroot --initrd --append-to-version=-tproxy.1.0
kernel_image
```

Wait a while for the kernel to compile. When it's done you'll have a linux-image package in `/usr/src`. Go ahead and install it:

```
cd /usr/src
dpkg -i linux-image-2.6.18*.deb
```

If the installation was successful, then we're almost done (with the kernel, anyway).

We want to load the tproxy modules by default, so add the following lines to the end of your `/etc/modules` file:

```
ip_tables
iptables_filter
ipt_TPROXY
ipt_tproxy
```

Reboot your machine into the new kernel. You can confirm the version by running `uname -a`. You can confirm that tproxy was installed by running:

```
dmesg | grep TPROXY
```

1.4.3 IPTables Compilation

The new version of the kernel has a patched version of iptables support, which requires that we recompile the iptables binaries to match.

First, move into your source directory:

```
cd /usr/src
```

Next, get the source for iptables:

```
apt-get source iptables
apt-get build-dep iptables
```

Move into the source directory:

```
cd iptables-1.3.6.0debian1
```

Patch the sources (note change of directory to inner iptables dir):

```
cd iptables
patch -p1 < ../../cttproxy-2.6.18-2.0.6/iptables/iptables-1.3-cttproxy.diff
chmod +x extensions/.tproxy-test
```

Additionally, we need to copy our patched kernel sources into the iptables tree so it can build against them. Change back to the root iptables source tree:

```
cd /usr/src/iptables-1.3.6.0debian1
```

Now move the default linux dir out of the way:

```
mv linux linux-orig
mkdir linux
```

And copy your linux files in:

```
for x in COPYING Makefile include net
do cp -a ../linux-source-2.6.18/$x linux/$x
done
```

Now you're ready to build. Substitute your e-mail address in the build command:

```
dpkg-buildpackage -rfakeroot -us -uc -b -myou@example.com
```

You'll end up with an iptables .deb file in your /usr/src directory. You can move into that directory and install the package:

```
cd /usr/src
dpkg -i iptables_1.3.6.0debian1-5_i386.deb
```

To confirm that everything is working correctly, try the following command:

```
iptables -t tproxy -L
```

If that doesn't give an error, then all of the modules are installed correctly, and you're ready to go.

1.4.4 Squid Compilation

For maximum performance, the Squid maintainers recommend that you compile your own version of Squid from scratch. Additionally, our "tproxy" setup is not included in Debian, so we must compile our own anyway.

Change into the source directory:

```
cd /usr/src
```

Fetch the source for squid, and the build dependencies:

```
apt-get source squid
apt-get build-dep squid
```

Move into the `squid-2.6.5` directory.

The Debian package-building system includes a `rules` file that dictates what configure options to use when building the package. We'll edit this file to include only the options that we want (you may wish to choose other options depending on your setup).

The Debian build has reasonable defaults for Linux (async-io, aufs, etc). We simply edit the rules file to remove code we don't need (mostly dealing with authentication methods). We also need to add tproxy support to the build.

In the `debian/rules` file, make the following changes:

- Add `--enable-linux-tproxy` to the list of build options
- Add `--enable-multicast-miss` to the list of build options
- Add `--disable-ident-lookups` to the list of build options (ident just slows us down)
- Remove `--enable-useragent-log` (don't need to log User Agent data)
- Remove `--enable-referer-log` (don't need to log referer)
- Remove `--enable-underscores` (don't allow illegal hostnames)
- Change the `--enable-auth` line to say `--enable-auth="basic,digest"` (Don't need NTLM auth)

Save the file when you're done.

We need to make the tproxy headers available to Squid for compilation. You can do this by running the following:

```
cp /usr/src/linux-source-2.6.18/include/linux/netfilter_ipv4/ip_tproxy.h \  
/usr/include/linux/netfilter_ipv4/
```

You must also install `libcap-dev` to get the capabilities file for compiling Squid (if you don't do this, you'll see "CAP_NET" errors):

```
apt-get install libcap-dev
```

Also, **be sure** to add `capability` to your `/etc/modules` file (Debian doesn't load capabilities by default).

You may wish to set certain compiler flags (*e.g.*, `CFLAGS='-O2 -march=pentium4'`) before the build to include any processor-specific options. Note that these packages may only work for the architecture they're compiled for!

Now, build the package! Substitute your e-mail address in the build command:

```
dpkg-buildpackage -rfakeroot -us -uc -b -myou@example.com
```

Once you've built the binary installation packages, you're ready to install them. They'll appear in the parent directory to your source folder, and there should be 4 Debian package files:

```
squid_2.6.5-6etch1_i386.deb  
squid-cgi_2.6.5-6etch1_i386.deb  
squidclient_2.6.5-6etch1_i386.deb  
squid-common_2.6.5-6etch1_all.deb
```

You'll need to install any dependencies manually. At the moment, this includes `lsb-base` (for squid) and a web server (for squid-cgi).

To install `lsb-base`:

```
sudo apt-get install lsb-base
```

For a web server, you can go with the tried-and-true apache. However, we want as lightweight a server as possible, so we opt for `thttpd`:

```
sudo apt-get install thttpd
```

You can install the Debian packages using `dpkg -i`:

```
sudo dpkg -i /usr/src/squid*.deb
```

Once this is done, you should have Squid installed, and ready to configure!

1.5 Configuration

At this point, you should have Squid installed on your system. You can confirm this by typing `squid -v` on the command line. You should get back Squid's version, along with any compile-time options.

If you don't have Squid yet, visit the previous section for information on compiling it from scratch, or install a packaged version from your OS vendor. For example, you could say this under Debian:

```
sudo apt-get install squid squidclient squid-cgi
```

Now that you've got Squid, it's time to configure it. Squid has a great deal of run-time options, which affect how it caches documents. We've tuned our config to work with our particular needs; the descriptions below explain the choices we've made. You should make changes as you see fit.

1.5.1 Bridge Setup

We configure our system as a network bridge, which means that it sits between two physical devices on our network and relays the packets between them. However, there's a twist: we intercept certain packets (those destined for port 80) and shunt them to Squid for processing.

You'll need two ethernet cards in your machine to bridge between (one "in" and one "out", as it were). You can use another card for a management IP address, or you can actually assign an address to the bridge itself and reach the machine just as you would a "real" interface.

In order to set up the bridge, we need to make a few tweaks to the system. First, we need to install some software that's necessary for setting up a bridge:

```
apt-get install bridge-utils
```

Next, edit `/etc/network/interfaces`. You should already have a stanza for a statically configured interface (*e.g.*, `eth0`). Keep the settings for the stanza, but replace the interface name with `br0`. Also, add the line `bridge_ports ethXXX ethYYY` to add them to the bridge. For example:

```
auto br0
iface br0 inet static
    bridge_ports eth0 eth1
    address 192.168.0.100
    netmask 255.255.255.0
    gateway 192.168.0.1
```

Additionally, if your setup is like ours you'll need to add some routing to the box so it knows where to send packets. Our Squid box sits just between our firewall/router and LAN. Thus, it needs to be told how to route packets to the LAN and packets to the outside world. We do this by specifying the firewall as the "gateway" in the `interfaces` file, and adding a static route for our LAN. Thus, you would add the following lines to `/etc/network/interfaces` in the `br0` stanza:

```
up route add -net 192.168.1.0/24 gw 192.168.1.1
down route del -net 192.168.1.1/24 gw 192.168.1.1
```

We'll need to tell the kernel that we're going to forward packets, so make sure the following are set in `/etc/sysctl.conf`:

```
net.ipv4.conf.default.rp_filter=1
net.ipv4.conf.default.forwarding=1
net.ipv4.conf.all.forwarding=1
```

Once you're all set, the easiest thing to do is reboot for the bridge config to take effect. The other settings should now be working also. `cat /proc/sys/net/ipv4/ip_forward` to confirm that the machine is in forwarding mode.

1.5.2 Disk Setup

Squid is going to need a place to store its data. On a single-disk system, this can be a directory or partition. On a multi-disk system (such as ours), we give entire disks over for Squid to use.

The listserver archives contain some debate about the best filesystem to use on the partition(s) for the Squid cache. From our reading, there appear to be 3 choices:

- `ext2` (a non-journaled Linux filesystem). It's very fast and lightweight, but the lack of journalling can mean long recovery times in the event of a crash.
- `reiserfs` (a journaled Linux filesystem). Also reported to be fast, and very efficient with directories containing large numbers of files (as with a cache).
- `xfs` (a journaled filesystem from SGI). Designed for high-demand multi-media storage (large files, large numbers of files).

We ruled out `ext2`, as we wanted a journaled system that could recover from crashes easily. That left the two journaled filesystems, `reiserfs` and `xfs`. Reiser

seems to have a loyal following, though users who have tried xfs report that it works very well (especially with the `epoll`, which is on by default in Linux and Squid 2.6). Because we use xfs as our standard Linux filesystem on our servers, we decided to use it for our cache as well.

Note that regardless of the filesystem selected, Squid users recommend using the `noatime` option for any cache partitions. This option prevents the system from updating the *last access time* for cache files, as it is unnecessary and just slows down the system.

Make sure that the directories are writable by your proxy user. On a Debian system, this means executing:

```
sudo chmod -R proxy:proxy /var/spool/squid
```

(Substitute your cache directory for `/var/spool/squid`.)

Also, if you add, remove, or change the locations of the cache directories, be sure to run `squid -z` to rebuild the cache directory structure.

1.5.3 Squid Configuration

The main configuration file is `/etc/squid/squid.conf`. It controls nearly every aspect of Squid's behavior, from memory consumption to ACLs to exceptions for caching rules.

We start with a stock Debian Squid configuration file and change it as necessary. Rather than describe the changes here, we comment the configuration file carefully. Simply search for the word **suffield** in the configuration file to find all of our changes.

Remember: we're building a bridged, intercepting (transparent) caching proxy with ad-blocking/monitoring. The options, numbers, and paths are specific to our setup and our machine. If you are building your own configuration, be careful to substitute values that work for your site.

[Download the Suffield Squid configuration file \(squid.conf\)](#)

1.5.4 TPROXY Rules

We've set our server up as a bridge, meaning that packets received on one network interface are sent out on the other interface. In this sense, the server "looks like" a straight wire to the rest of the network.

In order for squid to perform its work, we must divert packets passing over the bridge that we'd like to cache. Under Linux we can do this, which makes the

bridge not quite a bridge anymore (some people call it a "brouter", since it's more like a bridge combined with a router).

Our patched version of iptables is capable of doing everything we need. You should have already enabled ip forwarding via `sysctl` above. Once you have, you're ready to enable the redirection:

```
/sbin/iptables -t tproxy -A PREROUTING -p tcp -m tcp \  
--dport 80 -j TPROXY --on-port 81
```

(The command can be written on a single line by removing the backslash.)

A quick breakdown:

- `-t tproxy` means to affect the `tproxy` part of the routing tables. This is a special table that only exists if we've patched our kernel as described earlier in this document.
- `-A PREROUTING` means to "Add" a rule to the PREROUTING chain. This means that our rule will apply before the kernel applies its standard routing rules. This is what we want, since we're basically circumventing the standard rules.
- `-p tcp -m tcp` means to match only TCP traffic (not UDP or other protocols)
- `--dport 80` means to intercept traffic destined for port 80 (the port that HTTP traffic uses by default)
- `-j TPROXY` means to "Jump" to the TPROXY section in the kernel. `tproxy` will deal with maintaining the proxied connection's state, and do all the other "magic" for us.
- `--on-port 81` this is the local port on this machine to forward the request to. This should be the port that your squid instance is listening on, as defined in your `squid.conf` file. The line must read `http_port <port num> transparent tproxy`, with an actual number substituted for `<port num>`. We've selected a non-standard port of 81 so we can use the default squid port (3128) for manually-configured proxy hosts (you can't mix standard and transparent hosts on the same port). You can pick any port you want (8080, 8000, 666, etc), so long as it isn't in use. Just make sure that the port number matches in the squid config and your iptables rule.

That rule enables the redirection of packets to squid. You may want to add other options to it (for example, you probably only want to redirect packets

that are sourced from your trusted network range). Read the manual page for iptables for more information on these options.

To remove the rule and return to pure bridging, simply execute the same command, but substitute `-D` for `-A`. This says to "Delete" the rule, rather than add it:

```
/sbin/iptables -t tproxy -D PREROUTING -p tcp -m tcp \  
--dport 80 -j TPROXY --on-port 81
```

Read on in the next section for a way to automatically keep these rules in sync with the state of squid.

1.5.5 Watching Squid

We now have two independent processes (Squid and iptables) that must cooperate to make a functioning box. If we have squid running, but haven't enabled the iptables rules, nothing will be proxied. Meanwhile, if the iptables rules are enabled but squid isn't running, all web traffic will be blocked (making for unhappy users).

We've written a script to keep these items synchronized together. It runs all the time, periodically polling squid to make sure its running. If squid suddenly stops running, the script disables the iptables rules. Similarly, if squid returns, the rules are re-enabled. Finally, the script can also receive explicit commands to enable or disable the iptables rules (useful if you're planning to turn squid off; you can shut down the rules first to make for a smooth transition).

You can [download a copy of our squid-watcher script](#). It can be installed anywhere on the machine (we use `/usr/local/bin`).

You'll need to edit the script and check the variables at the top. They define how frequently to poll squid, and how to alert you if something stops working (syslog or e-mail).

Also, the script defines two "hooks" where you can specify the commands to run when squid starts and stops. This is where you should put your iptables rules (our rules are there; you can modify them to suit your needs).

To set it up to run, you'll need to add a line to `/etc/inittab` that looks like this:

```
sq:2345:respawn:/path/to/squid-watcher poll < /dev/null > /dev/null 2>&1
```

Make sure to set the correct path to the script.

Once everything is ready to go, reload INIT (we assume init is PID 1 on your machine; use `ps -e | grep init` to confirm):

```
kill -HUP 1
```

You should see a message in the system logs, or via e-mail (depending on how you configured logging) confirming that the script has started.

If squid is running, you can force the script to enable your rules by running:

```
squid-watcher start
```

To disable your rules, you can run:

```
squid-watcher stop
```

Finally, you can ask the script about its current state by running:

```
squid-watcher info
```

You'll see a message in the logs telling you what the current state is. It will be one of the following:

```
RUNNING - Squid is running, and the script's rules are enabled
STOPPED - Script's rules are disabled (squid's status is not checked)
BROKEN - Script tried to enable rules, but squid is not running
```

The script will automatically move between the `RUNNING` and `BROKEN` states, depending on the state of squid. The `STOPPED` state prevents the script from changing anything (useful when you know that you're turning squid off, and don't want the script to recover for you).

You should add commands to your squid init scripts (see below) to make sure that you enable and disable the script when appropriate.

1.5.6 Initialization Scripts

These commands can be added to the Squid init scripts, which live in `/etc/init.d`. You can see the changes we've made by [downloading our Squid init script](#).

Basically, whenever we tell squid to start, we also tell our `squid-watcher` script to start as well. Similarly, just before we stop squid we tell `squid-watcher` to stop monitoring (so it doesn't freak out when squid terminates). Other than that, it's pretty much the stock config.

Other minor changes

We've also tweaked our init script to perform other minor tuning. We set the "swappiness" of the VM in the kernel to prevent the machine from swapping too aggressively (we'd rather use the memory for caching!).

You can do this by adding:

```
vm.swappiness=10
```

to `/etc/sysctl.conf`

1.6 Cache Manager

Squid includes a `cachemgr` CGI binary that allows you to query a running Squid instance for statistics. These stats can help you determine if Squid is running efficiently.

You do not need to install `cachemgr` on the same machine as Squid (for example, you may wish to install it on a standard web server). We choose to run `cachemgr` on the local machine in a lightweight HTTP server such as `thttpd`.

1.6.1 Configuration

The `cachemgr` program has its own configuration file in `/etc/squid/cachemgr.conf`. The only configurable option is a list of server addresses, ports, and descriptions. Enter the addresses of the cache(s) you wish to monitor.

1.6.2 Security

By default, Squid does not allow `cachemgr` to perform destructive operations (shutdowns, reloads, *etc.*). Even if you don't enable these features, however, you may wish to restrict access to `cachemgr`, either through firewall rules or web server access controls.

1.7 Squid Statistics

To keep track of how well your Squid installation is doing, it's helpful to analyze the logs that it produces. Numerous log analyzers exist for Squid; some are text-only, others HTML, and others are graphical.

At Suffield, we've settled on an analyzer called Calamaris:

<http://cord.de/tools/squid/calamaris/>

The software has few requirements, is written in Perl, and can produce text and HTML reports. The latest beta (V2.99.4.0 as of this writing) also supports creating graphs in the HTML version.

1.7.1 Downloading and Installing Calamaris

Debian includes a version of Calamaris in its software repository. Thus, you can install it directly using:

```
apt-get install calamaris libgd-graph-perl
```

For other platforms, check the software repository to see if a packaged version is available. Otherwise, you may download the script from the website listed above.

1.7.2 Configuring Calamaris

The Calamaris configuration file is really just a large Perl file that gets included at runtime. You may set all the options relating to how the script produces output in this file. Our version of the config file is commented with all the changes we've made, and you can [download it from our web server](#). All of our changes are commented with the word "suffield".

Additionally, Debian provides another file called `cron.conf` which defines the reports that get generated, where they are stored (or e-mailed), and how often to generate the reports. You may [download our version of this config file](#) to see how we've configured it.

Finally, Debian uses a script launched from `cron` to execute Calamaris with the proper options. We've edited this script to make it work more the way we want. There are only two major changes:

1. Modify the report format to produce frameset-HTML with embedded graphs.
2. Modify the output file and directory names to be specified by date (rather than just overwriting the logs each time).

The changes are well-commented and pretty straightforward. [Download our version of the cron script](#) to see the changes we've made.

1.7.3 Running Calamaris

Calamaris is automatically executed by `cron` every morning. The output is mailed or dumped to a file (or both) when processing completes.

If you choose to output to an HTML file, you should configure the script to drop the reports directly into your web server directory. The files will be made available immediately via your web server.