

The Squid and the Blowfish

Author: [Daniele Mazzocchio](#)

Applies to: OpenBSD 4.7

Last update: Jun 11, 2010

Latest version: <http://www.kernel-panic.it/openbsd/proxy/>

Table of Contents

1. Introduction.....	2
2. Squid.....	3
2.1 Installation.....	3
2.2 Base configuration.....	3
2.3 Starting Squid.....	6
3. Further Squid configuration.....	8
3.1 More on Access Control Lists.....	8
3.2 Http-accelerator mode (reverse proxy).....	11
3.3 Transparent caching.....	11
3.4 SNMP.....	12
4. Content filtering with SquidGuard.....	14
4.1 Installation.....	14
4.2 Configuration.....	14
5. Virus scanning with SquidClamav.....	17
5.1 Installation.....	17
5.2 Configuration.....	17
6. Ad Zapping with AdZapper.....	20
6.1 Installation.....	20
6.2 Configuration.....	20
7. Appendix.....	22
7.1 Tunneling through Squid.....	22
7.1.1 Server-side configuration.....	22
7.1.2 Client-side configuration.....	23
7.2 References.....	24
7.3 Bibliography.....	24

1. Introduction

We have grown so much accustomed to Internet access on our work computers, that we can hardly imagine what people ever did all day long on their workplace before!

By providing access to a virtually endless amount of information, the Internet has quickly turned into an essential working tool. So essential that most companies can't do without it anymore. But besides providing a huge amount of information, the Internet has also turned into the main virus vehicle (together with e-mail) and doesn't exclusively provide content in line with corporate policies. That's why a proxy server is often as necessary as the Internet connection itself.

The main benefits of web proxying are:

- content filtering: the proxy can be configured to filter out virus files, ad banners and requests to unwanted websites;
- network bandwidth conservation: cached pages are served by the proxy itself, thus saving bandwidth and offering faster access times;
- authentication: Internet access can be authorized (and filtered) based on username/password, IP address, domain name and much more.

The following is the list of the pieces of software we will use:

OpenBSD

a robust, security-oriented operating system, with *“only two remote holes in the default install, in a heck of a long time!”*;

Squid

a *“caching proxy for the Web supporting HTTP, HTTPS, FTP, and more”*;

SquidGuard

a *“combined filter, redirector and access controller plugin for Squid”*;

ClamAV

a fast and easy-to-use open-source virus scanner;

SquidClamav

an *“antivirus redirector for Squid proxy based on the Awards winnings ClamAv anti-virus toolkit”*;

AdZapper

a *“redirector for squid that intercepts advertising (banners, popup windows, flash animations, etc), page counters and some web bugs (as found)”*.

The choice of using free software prevented me from using [DansGuardian](#), an Open Source web content filter, running on many OSes and filtering the actual content of pages based on many methods including phrase matching, PICS filtering and URL filtering. Fine and dandy, but it is *not* free for [commercial use](#).

A good knowledge of OpenBSD is assumed, since we won't delve into system management topics such as OS installation and base configuration, packages/ports installation or PF syntax.

2. Squid

Squid is a “a full-featured HTTP/1.0 proxy” and it “offers a rich access control, authorization and logging environment to develop web proxy and content serving applications”.

2.1 Installation

Let's start with the location of the cache server in the network: according to the [documentation](#), the most suitable place is in the DMZ; this should keep the cache server secure while still able to peer with other, outside, caches (such as the ISP's).

The [documentation](#) also recommends setting a DNS name for the cache server (such as "cache.mydomain.tld" or "proxy.mydomain.tld") as soon as possible: “a simple DNS entry can save many hours further down the line. Configuring client machines to access the cache server by IP address is asking for a long, painful transition down the road”.

Squid installation is as simple as it can be; you only have to add the Squid package. Available flavors are "ldap" (allowing for LDAP authentication) and "snmp" (including [SNMP](#) support).

```
# export PKG_PATH=/path/to/your/favourite/OpenBSD/mirror
# pkg_add squid-x.x.STABLExx-snmppkg.tgz
squid-x.x.STABLExx-snmppkg: complete
--- squid-x.x.STABLExx-snmppkg -----
NOTES ON OpenBSD POST-INSTALLATION OF SQUID x.x

The local (OpenBSD) differences are:
configuration files are in           /etc/squid
sample configuration files are in    /usr/local/share/examples/squid
error message files are in          /usr/local/share/squid/errors
sample error message files are in   /usr/local/share/examples/squid/errors
icons are in                         /usr/local/share/squid/icons
sample icons are in                 /usr/local/share/examples/squid/icons
the cache is in                     /var/squid/cache
logs are stored in                  /var/squid/logs
the uid squid runs as is            _squid:_squid

Please remember to initialize the cache by running "squid -z" before
trying to run Squid for the first time.

You can also edit /etc/rc.local so that Squid is started automatically:

if [ -x /usr/local/sbin/squid ]; then
    echo -n ' squid';           /usr/local/sbin/squid
fi

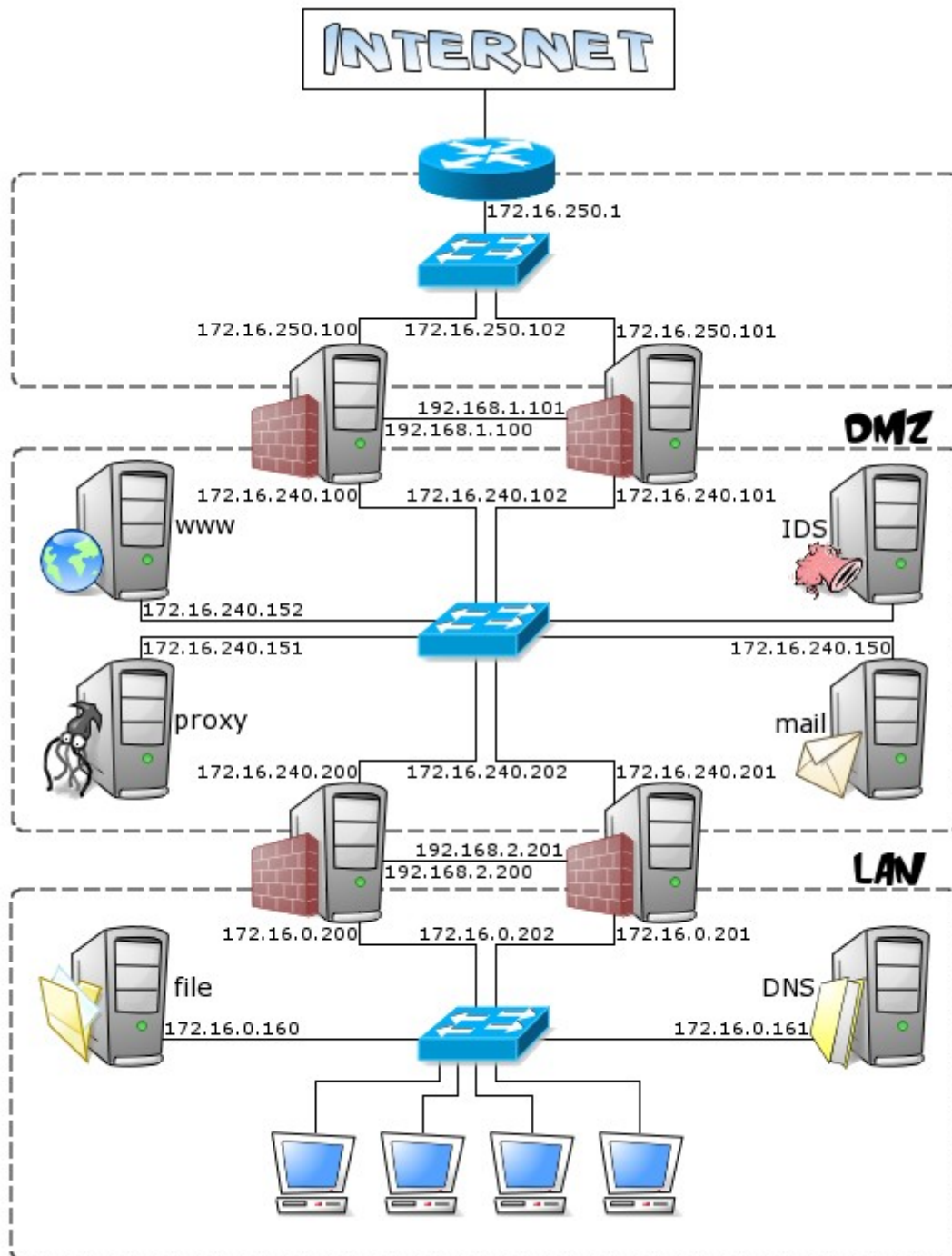
#
```

2.2 Base configuration

Squid configuration relies on several dozens of parameters, and thus can quickly turn into a very tricky task. Therefore, the best approach is probably starting with a very basic configuration and then tweaking the options, one by one, to meet your specific needs, while still making sure that everything keeps working as expected.

Actually, only a few parameters need to be set to get Squid up and running (theoretically, you could even run Squid with an empty configuration file): for all the options you don't explicitly set, the default values are assumed. Anyway, at least one setting must certainly be changed: the default configuration file denies access to all browsers; and this may sound a bit ...too strict!

Our first configuration will be very simple: we will place our proxy server in the DMZ (172.16.240.0/24, below is the network layout) and allow only requests from the LAN (172.16.0.0/24). No ISP's parent proxy is taken into account.



The main Squid configuration file is `/etc/squid/squid.conf`. Let's have a look at it.

The `http_port` option sets the port(s) that Squid will listen on for incoming HTTP requests. There are three forms: port alone (e.g. "`http_port 3128`"), hostname with port (e.g. "`http_port proxy.kernel-panic.it:3128`"), and IP address with port (e.g. "`http_port 172.16.240.151:3128`"); you can specify multiple socket addresses, each on a separate line. If your Squid machine is multi-homed and directly accessible from the internet, it is strongly recommended that you force Squid to bind the socket to the internal address. This way, Squid will only be visible from the internal network and won't proxy the whole world! Squid's default HTTP port is 3128, but many administrators prefer using a port which is easier to remember, such as 8080.

```
http_port 3128
```

The `cache_dir` parameter allows you to specify the path, size and depth of the directories where the cache swap files will be stored. Squid allows you to have multiple `cache_dir` tags in your config file.

```
cache_dir ufs /var/squid/cache 100 16 256
```

The above line sets the cache directory pathname to `/var/squid/cache`, with a size of 100MB and 16 first-level subdirectories, each containing 256 second-level subdirectories. The cache directory must exist and be writable by the Squid process and its size can't exceed 80% of the whole disk. For further details, please refer to the [documentation](#).

The `cache_mgr` parameter contains the e-mail address of the Squid administrator, which will appear at the end of the error pages; e.g.:

```
cache_mgr webmaster@kernel-panic.it
```

The `cache_effective_user` and `cache_effective_group` options, allow you to set the UID and GID Squid will drop its privileges to once it has bound to the incoming network port. The package installation has already created the `_squid` user and group.

```
cache_effective_user _squid
cache_effective_group _squid
```

The `ftp_user` option sets the e-mail address that Squid will use as the password for anonymous FTP login. It's a good practice to use an existing address:

```
ftp_user webmaster@kernel-panic.it
```

The following options set the paths to the log files; the format of the access log file, which logs every request received by the cache, can be specified by using a `logformat` directive (please refer to the [documentation](#) for a detailed list of the available format codes):

```
# Define the access log format
logformat squid %ts.%03tu %6tr %>a %Ss/%03Hs %<st %rm %ru %un %Sh/%<A %mt
# Log client request activities ('squid' is the name of the log format to use)
access_log /var/squid/logs/access.log squid

# Log information about the cache's behavior
cache_log /var/squid/logs/cache.log
# Log the activities of the storage manager
cache_store_log /var/squid/logs/store.log
```

And now we come to one of the most tricky parts of the configuration: Access Control Lists. The simplest way to restrict access is to only accept requests from the internal network. Such a basic access control can be enough in small networks, especially if you don't wish to use features like username/password authentication or URL filtering.

ACLs are usually split into two parts: `acl` lines, starting with the `acl` keyword and defining classes, and `acl` operators, allowing or denying requests based on classes. Acl-operators are checked from top to bottom and the first matching wins. Below is a very basic ruleset:

```
# Classes
acl all src all # Any IP address
acl localhost src 127.0.0.0/8 # Localhost
acl lan src 172.16.0.0/24 # LAN where authorized clients reside
acl manager proto cache_object # Cache object protocol
acl to_localhost dst 127.0.0.0/8 # Requests to localhost
```

```

acl SSL_ports      port 443          # https port
acl Safe_ports     port 80 21 443     # http, ftp, https ports
acl CONNECT        method CONNECT    # SSL CONNECT method

# Only allow cachemgr access from localhost
http_access allow  manager localhost
http_access deny   manager

# Deny requests to unknown ports
http_access deny   !Safe_ports

# Deny CONNECT to other than SSL ports
http_access deny   CONNECT !SSL_ports

# Prevent access to local web applications from remote users
http_access deny   to_localhost

# Allow access from the local network
http_access allow  lan

# Default deny (this must be the last rule)
http_access deny   all

```

2.3 Starting Squid

Now our cache server is almost ready for a first run, just one last step to go. We first need to create the cache-swap directories where Squid will store cached pages. The "squid -z" command will create all the required directories, according to the `cache_dir` parameter in `squid.conf` (see [above](#)), as the user and group specified by the `cache_effective_user` and `cache_effective_group` parameters.

```

# /usr/local/sbin/squid -z
2009/10/30 18:04:35| Creating Swap Directories
#

```

We are now ready to start Squid. Starting it in debug mode (`-d 1` flag) and in foreground (`-N` flag) will make it easier to see if everything is working fine.

```

# /usr/local/sbin/squid -d 1 -N
2009/10/30 18:05:19| Starting Squid Cache version 2.7.STABLE6 for i386-unknown-
openbsd4.6...
[ ... ]
2009/10/30 18:05:19| Accepting HTTP connections at 0.0.0.0, port 3128, FD 10.
2009/10/30 18:05:19| Accepting ICP messages at 0.0.0.0, port 3130, FD 11.
2009/10/30 18:05:19| Accepting SNMP messages on port 3401, FD 12.
2009/10/30 18:05:19| WCCP Disabled.
2009/10/30 18:05:19| Ready to serve requests.
2009/10/30 18:05:22| Done scanning /var/squid/cache (0 entries)
2009/10/30 18:05:22| Finished rebuilding storage from disk.
2009/10/30 18:05:22|           0 Entries scanned
2009/10/30 18:05:22|           0 Invalid entries.
2009/10/30 18:05:22|           0 With invalid flags.
2009/10/30 18:05:22|           0 Objects loaded.
2009/10/30 18:05:22|           0 Objects expired.
2009/10/30 18:05:22|           0 Objects cancelled.
2009/10/30 18:05:22|           0 Duplicate URLs purged.
2009/10/30 18:05:22|           0 Swapfile clashes avoided.
2009/10/30 18:05:22| Took 5.1 seconds ( 0.0 objects/sec).
2009/10/30 18:05:22| Beginning Validation Procedure
2009/10/30 18:05:22| Completed Validation Procedure

```

```
2009/10/30 18:05:22| Validated 0 Entries
2009/10/30 18:05:22| store_swap_size = 0k
2009/10/30 18:05:22| storeLateRelease: released 0 objects
```

Once you get the "Ready to serve requests" message, you should be able to use the cache server. Once it is up and running, Squid reads the cache store: the first time you should see all zeros, as above, because the cache store is empty.

Now, to make sure everything is working fine, we will configure our browser to use our fresh new proxy and we will try to access our favourite web site. In the `/var/squid/logs/access.log` file, you should see something like:

```
/var/squid/logs/access.log
```

```
1242419601.435 6735 172.16.0.13 TCP_MISS/200 11810 GET http://www.kernel-panic.it/
- DIRECT/62.149.140.23 text/html
1242419849.536 14 172.16.0.13 TCP_HIT/200 11820 GET http://www.kernel-panic.it/
- NONE/- text/html
[...]
```

For a detailed description of each field in the `access.log` file, please refer to the [documentation](#). Anyway, `TCP_MISS` means that the requested page wasn't stored in the cache (either it was not present or it had expired); `TCP_HIT`, instead, means that the page was served from the cache. The second field is the time (in milliseconds) that Squid took to service the request: as you can see, it is much shorter when the page is cached. The page size is the fifth field: cached pages may be a little larger because of the extra headers added by Squid.

If everything is working fine, we can stop Squid:

```
# /usr/local/sbin/squid -k shutdown
```

and configure the system to start it on boot.

```
/etc/rc.local
```

```
if [ -x /usr/local/sbin/squid ]; then
    echo -n ' squid'
    /usr/local/sbin/squid
fi
```

You may also wish to start Squid through the `RunCache` script, which automatically restarts it on failure and logs both to the `/var/squid/squid.out` file and to `syslog`. Just remember to background it with an `&`, or it will hang the system at boot time.

3. Further Squid configuration

In many cases, the basic configuration we've seen in the [previous chapter](#) can be sufficient for accelerating web access and protecting the network, but Squid can do much more. Below are a just few of the many things Squid can do.

3.1 More on Access Control Lists

Though most people implement only very basic access control, Squid's access system is very powerful and flexible, allowing for in-depth filtering of access to cache resources. So far we have mainly dealt with ACLs that filter based on source IP address or destination port, but there are many other ACL types. In this paragraph, we will take a brief look at the main ones, just to get an idea of what Squid ACLs can do; for a more detailed and comprehensive description of Squid ACLs, please refer to the [documentation](#).

A Squid ACL is made up of at least four fields: the `acl` keyword, followed by a (possibly descriptive) unique name, the ACL type and one or more decision strings. Thus, the overall syntax of Squid ACLs looks like:

```
acl name type (string|"filename") [string2] [string3] ["filename2"]
```

An ACL containing multiple decision strings will return true if any of the decision strings matches (i.e. decision strings are ORed together). To avoid cluttering the configuration file with hundreds of ACL lines, you can specify the full pathname of a file (in double quotes) containing the decision strings one per line.

Listed below are the most commonly used ACL types:

Source/Destination IP address

Filtering based on source IP address (`src` type) or destination IP address (`dst` type). Both the traditional "IP/Netmask" and CIDR "IP/Bits" notations are allowed. E.g.:

```
# "Traditional" notation
acl myNet1 src 192.168.0.0/255.255.255.0

# Address range with CIDR notation
acl myNet2 src 172.16.0.0-172.16.2.0/24

# Filtering on destination address
acl badNet dst 10.0.0.0/24
```

Source/Destination Domain

Squid can allow/deny requests to or from specific domains (`dstdomain` and `srcdomain` types, respectively). If you want to deny access to a site, don't forget to also deny access to its IP address, or the rule will be easily bypassed. E.g.:

```
# Match a specific site
acl badDomain dstdomain forbidden.site

# Match the IP address of "forbidden.site"
acl badDomainIP dst 1.2.3.4
```

Regular expressions can also be used for checking the source domain (`srcdom_regex` type) and destination domain (`dstdom_regex` type) of a request. E.g.:

```
# Match domains containing the word "sex" and a ".com" TLD (the match is case
# insensitive because of the '-i' flag)
acl badSites dstdom_regex -i sex.*\.com$
```

Words in the requested URL

Squid can use regular expressions to filter URLs matching specific patterns (`url_regex` type); if you don't care about the URL-type and the hostname, you can use the `urlpath_regex` type instead.

```
# Match the most common video files extensions
acl movies urlpath_regex -i \.avi$ \.mpg$ \.mpeg$ \.wmv$ \.asf$ \.mov$

# Match JPG images from URLs containing the word "sex"
acl sexImg url_regex -i sex.*\.jpg$
```

Current day/time

Squid can allow/deny access to specific sites by time. The syntax is:

```
acl name time [day-list] [start_hour:minute-end_hour:minute]
```

where `day-list` is a list of single characters representing the days that the acl applies to (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday). E.g.:

```
acl workhours time MTWHF 08:00-18:00
acl weekend time SA
acl morning time 07:00-13:00
```

Destination port

Squid can filter based on destination ports. E.g.:

```
acl SSL_ports port 443 563
acl Safe_ports port 80 21 443 563 70 210 280 488 591 777 1024-65535
```

Protocol (FTP, HTTP, SSL)

The `proto acl` type allows Squid to allow/deny access based on the request protocol. E.g.:

```
acl www proto HTTP SSL
acl ftp proto FTP
```

Method (HTTP GET, POST or CONNECT)

The method ACL type allows you to restrict access based on the request HTTP method, i.e. GET (used for downloading), POST (used for uploading) and CONNECT (used for SSL data transfers). E.g.:

```
# Deny CONNECT to other than SSL ports
acl connect method CONNECT
http_access deny connect !SSL_ports
```

It is very important that you stop CONNECT type requests to non-SSL ports. The CONNECT method allows data transfer in any direction at any time, regardless of the transport protocol used. As a consequence, a malicious user could [telnet\(1\)](#) to a (very) badly configured proxy, enter something like:

```
$ telnet bad.proxy.tld 3128
Trying 1.2.3.4...
Connected to bad.proxy.tld.
Escape character is '^]'.
CONNECT telnet.server.tld:23 HTTP/1.1
```

and end up connected to the remote server, as if the connection was originated from the proxy.

Browser type

The browser acl type allows you to specify a regular expression that can be used to allow/deny access based on the User-Agent header. E.g.:

```
# Deny access to MS Internet Explorer
acl MSIE browser MSIE
http_access deny MSIE
```

Username/Password pair

User authentication allows you to track Internet usage and collect per-user statistics. The simplest authentication scheme is the basic scheme, with username/password pairs stored in a file. To create this file, you can use the [htpasswd\(1\)](#) command:

```
# /usr/bin/htpasswd -c /etc/squid/squid.passwd danix
New password: dAn1x
Re-type new password: dAn1x
Adding password for user danix
#
```

Authentication parameters are set using the `auth_param` tag; then, to actually activate authentication, you need to make use of ACLs based on login name in `http_access` (`proxy_auth` or `proxy_auth_regex`) or `external_acl_type` with `%LOGIN` used in the format tag. E.g.:

```
# Configure traditional (basic) proxy authentication
auth_param basic program /usr/local/libexec/ncsa_auth /etc/squid/squid.passwd

# Number of authenticator processes to spawn
auth_param basic children 5

# Realm to be reported to the client
auth_param basic realm Squid proxy-caching web server

# Usernames are case insensitive
auth_param basic casesensitive off

# Credentials time to live
auth_param basic credentialsttl 12 hours

# Using REQUIRED will accept any valid username
acl AUTH proxy_auth REQUIRED

# Don't require authentication to localhost
http_access allow localhost

# Only allow authenticated requests coming from the LAN
http_access allow AUTH lan

# Default deny
http_access deny all
```

SNMP Community

Squid can restrict SNMP queries based on the requested SNMP community. E.g.:

```
# Address of the cache administrator
acl snmpManager src 172.16.0.100

# Non-sensitive information
acl SNMPPublic snmp_community public
```

```
# Allow any request from the cache administrator
snmp_access allow snmpManager

# Clients on the LAN can only query non-sensitive information
snmp_access allow SNMPPublic lan

# Default deny
snmp_access deny all
```

3.2 Http-accelerator mode (reverse proxy)

According to the [documentation](#), enabling Squid's Accelerator Mode can be useful only in a limited set of circumstances:

- accelerating a slow server;
- replacing a combination cache/web server with Squid;
- transparent caching;
- protecting an insecure web server.

Besides these cases, enabling the accelerator mode is strongly discouraged. The configuration is very simple; below is a sample configuration of a Squid server accelerating requests to a slow web server.

```
/etc/squid/squid.conf
```

```
# In accelerator mode, Squid usually listens on the standard www port
http_port 80 accel vhost

# Do the SSL work at the accelerator level. To create the certificates, run:
# openssl req -x509 -newkey rsa:2048 -keyout squid.key -out squid.crt \
# -days 365 -nodes
https_port 443 cert=/etc/ssl/squid.crt key=/etc/ssl/private/squid.key

# Accelerated server address and port
cache_peer 172.16.1.217 parent 80 0 no-query originserver

# Do not rewrite 'Host:' headers
url_rewrite_host_header off
# Process multiple requests for the same URI as one request
collapsed_forwarding on

# Allow requests when they are to the accelerated machine AND to the
# right port
acl webSrv dst 172.16.1.217
acl webPrt port 80
acl all src 0.0.0.0/0.0.0.0
http_access allow webSrv webPrt
http_access allow all
always_direct allow webSrv
```

3.3 Transparent caching

Transparent caching means having a filtering device, such as a router or a firewall, silently redirecting web traffic to the cache server. Clients ignore the presence of the proxy between them and the web server and think they're talking directly to the server.

As a consequence, transparent caching doesn't require any configuration on the client side, thus making maintenance much easier and faster. On the other hand, however, a transparently intercepting proxy can't use authentication or transparently proxy the HTTPS protocol.

Before configuring Squid, we will need to enable web traffic redirection on our [firewalls](#) (the involved firewalls are those between the LAN, where clients reside, and the DMZ, where the cache server is

placed). Below are some sample rules for the [pf.conf\(5\)](#) file:

```
/etc/pf.conf

[...]
# LAN interface
lan_if      = rl1

# Cache server and port
cache_srv   = proxy.kernel-panic.it
cache_port  = 3128

# Transparently redirect web traffic to the cache server
pass in on $lan_if proto tcp from $lan_if:network to any port www \
    rdr-to $cache_srv port $cache_port
[...]
```

Squid configuration is quite simple:

```
/etc/squid/squid.conf

# Port on which connections are redirected
http_port 3128 transparent
```

3.4 SNMP

SNMP is a set of protocols for network management and monitoring. If you [installed](#) the “snmp” flavor of the Squid package, the proxy will be able to serve statistics and status information via SNMP.

SNMP configuration is rather simple:

```
/etc/squid/squid.conf

# By default, Squid listens for SNMP packets on port 3401, to avoid conflicting
# with any other SNMP agent listening on the standard port 161.
snmp_port 3401

# Address to listen on (0.0.0.0 means all interfaces)
snmp_incoming_address 0.0.0.0

# Address to reply on (255.255.255.255 means the same as snmp_incoming_address)
# Only change this if you want to have SNMP replies sent using another address
# than where Squid listens for SNMP queries.
# snmp_incoming_address and snmp_outgoing_address can't have the same value
# since they both use port 3401.
snmp_outgoing_address 255.255.255.255

# Configuring access control is strongly recommended since some SNMP
# information is confidential
acl all src 0.0.0.0/0.0.0.0
acl lan src 172.16.0.0/24
acl snmpManager src 172.16.0.100
acl publicCommunity snmp_community public
snmp_access allow snmpManager
snmp_access allow publicCommunity lan
snmp_access deny all
```

You can test whether SNMP is working with the [snmpwalk](#) program (snmpwalk is part of the [NET-SNMP](#) project). E.g.:

```
# snmpwalk -c public -v 1 proxy.kernel-panic.it:3401 .1.3.6.1.4.1.3495.1.1
SNMPv2-SMI::enterprises.3495.1.1.1.0 = INTEGER: 356
SNMPv2-SMI::enterprises.3495.1.1.2.0 = INTEGER: 744
```

```
SNMPv2-SMI::enterprises.3495.1.1.3.0 = Timeticks: (540791) 1:30:07.91  
#
```

Please refer to the [documentation](#) for a detailed explanation of the output from the `snmpwalk` command.

4. Content filtering with SquidGuard

[SquidGuard](#) is a “combined filter, redirector and access controller plugin for Squid”. We will use it to block access to specific categories of unwanted sites, based on IP addresses, URLs and regular expressions. SquidGuard comes with a very comprehensive list of commonly-banned web sites, divided into categories such as “porn”, “drugs”, “ads” and so on, making configuration rather simple and fast.

4.1 Installation

SquidGuard is available through OpenBSD's [packages and ports system](#) and requires the installation of the following packages:

- db-x.x.x.tgz
- squidGuard-x.x.x.tgz

The installation places a copy of the blacklists tarball (blacklists.tar.gz) in /usr/local/share/examples/squidguard/dest/. We will extract it into the /var/squidguard/db directory:

```
# cd /usr/local/share/examples/squidguard/dest/
# mkdir -p /var/squidguard/db
# tar -zxvC /var/squidguard/db -f blacklists.tar.gz
[...]
```

4.2 Configuration

SquidGuard's configuration file is /etc/squidguard/squidguard.conf; it is logically divided into six sections (please refer to the [documentation](#) for a more in-depth look at squidGuard's configuration options):

Path declarations

Specify the path to the logs and blacklists directories:

```
logdir /var/squidguard/log
dbhome /var/squidguard/db
```

Time space declarations

SquidGuard allows you to have different access rules based on time and/or date. A short example will probably best illustrate the flexibility of these rules.

```
time workhours {
    weekly mtwhf 08:00-18:00
}

time night {
    weekly * 18:00-24:00
    weekly * 00:00-08:00
}

time holidays {
    date *.01.01 # New Year's Day
    date *.05.01 # Labour Day
    date *.12.24 12:00-24:00 # Christmas Eve (short day)
    date *.12.25 # Christmas Day
    date *.12.26 # Boxing Day
}
```

Source group declarations

SquidGuard allows you to filter based on source IP address, domain and user (users credentials are passed by Squid along with the URL); e.g.:

```
src admin {
  ip      172.16.0.12          # The administrator's PC
  domain  lan.kernel-panic.it # The LAN domain
  user    root administrator  # The administrator's login names
}

src lan {
  ip      172.16.0.0/24       # The internal network
  domain  lan.kernel-panic.it # The LAN domain
}
```

Destination group declarations

One of the main features of SquidGuard is certainly its ability to filter based on destination address or domain. And this is where the pre-built databases we extracted before come in handy. The `domainlist` parameter specifies the path to a file containing a list of domain names (later on, we will see how to create the db files to speed up SquidGuard startup time): this must be a relative path rooted in the directory specified by the `dbhome` parameter. Similarly, the `urllist` and `expressionlist` parameters specify the (relative) path to files containing a list of URLs and regular expressions respectively. E.g.:

```
dest porn {
  domainlist  blacklists/porn/domains
  urllist     blacklists/porn/urls
  expressionlist blacklists/porn/expressions
  # Logged info is anonymized to protect users' privacy
  log anonymous dest/porn.log
}
```

Access control rule declarations

Finally, we can combine all the previous rules to build Access Control Lists:

```
acl {
  admin within workhours {
    # The following rule allows everything except porn, drugs and
    # gambling sites during work hours. '!' is the NOT operator.
    pass !porn !drugs !gambling all
  } else {
    # Outside of work hours drugs and gambling sites are still blocked.
    pass !drugs !gambling all
  }
  lan {
    # The built-in 'in-addr' destination group matches any IP address.
    pass !in-addr !porn !drugs !gambling all
  }
  default {
    # Default deny to reject unknown clients
    pass none
    redirect http://www.kernel-panic.it/error.html&ip=%a&url=%u
  }
}
```

The `redirect` rule declares the URL where to redirect users requesting blocked pages. SquidGuard can include some useful information in the URL by expanding the following macros:

- `%a`: the IP address of the client.
- `%n`: the domain name of the client or "unknown" if not available.

- %i: the user ID or "unknown" if not available.
- %s: the matched source group or "unknown" if no groups were matched.
- %t: the matched destination group or "unknown" if no groups were matched.
- %u: the requested URL.
- %p: the path and the optional query string of %u but without the leading "/".
 - %%: a single "%".

Now that squidGuard is configured, we can build the Berkeley DB files for domains, URLs and regular expressions with the command:

```
# squidGuard -u -C all
# chown -R _squid /var/squidguard/
```

You can test that squidGuard configuration is working properly by simulating some Squid requests from the command line; squidGuard expects a single line on stdin with the following format (empty fields are replaced with "-"):

```
URL client_ip/fqdn user method urlgroup
```

and returns the configured redirect URL (if the site is blocked) or an empty line; for example:

```
# echo "http://www.blocked.site 1.2.3.4/- user GET -" | squidGuard \
> -c /etc/squidguard/squidguard.conf -d
[ ... ]
2008-12-14 09:57:04 [27349] squidGuard ready for requests (1197622624.065)
http://www.kernel-panic.it/error.html&ip=1.2.3.4&url=http://www.blocked.site
1.2.3.4/- user GET
2008-12-14 09:57:04 [27349] squidGuard stopped (1197622624.067)
# echo "http://www.good.site 1.2.3.4/- user GET -" | squidGuard \
> -c /etc/squidguard/squidguard.conf -d
[ ... ]
2008-12-14 10:30:24 [12046] squidGuard ready for requests (1197624624.421)
2008-12-14 10:30:24 [12046] squidGuard stopped (1197624624.423)
```

If everything is working as expected, we can configure Squid to use squidGuard as the redirector, by editing a few parameters in the `/etc/squid/squid.conf` file.

```
/etc/squid/squid.conf
```

```
# Path to the redirector program
url_rewrite_program /usr/local/bin/squidGuard

# Number of redirector processes to spawn
url_rewrite_children 5

# To prevent loops, don't send requests from localhost to the redirector
url_rewrite_access deny localhost
```

and reload Squid configuration:

```
# squid -k reconfigure
```

5. Virus scanning with SquidClamav

[SquidClamav](#) is a “*ClamAV antivirus redirector for Squid*”. It will help us filter out malicious software from web traffic.

5.1 Installation

We already covered the installation procedure of the [Clam AntiVirus](#) in a [previous document](#), so we won't dwell on this topic now and proceed directly to the installation of SquidClamav. We will assume that ClamAV resides on the same machine as Squid, though you may wish to create a separate antivirus server, possibly serving both the cache and the mail server.

SquidClamav relies on the [cURL](#) library to download the files to scan, so we need to add the following packages first:

- libiconv-x.x.tgz
- gettext-x.x.x.tgz
- libidn-x.x.tgz
- curl-x.xx.x.tgz

Then we can [download](#), extract and compile the SquidClamav tarball:

```
$ tar -zxvf squidclamav-x.x.tar.gz
[...]
$ cd squidclamav-x.x
$ env LDFLAGS=-L/usr/local/lib/ CPPFLAGS=-I/usr/local/include/ ./configure
[...]
$ make
[...]
$ su
Password:
$ make install
[ ... ]
# cp etc/squidclamav.conf /etc/
# touch /var/log/squidclamav.log
# chown _squid /var/log/squidclamav.log
```

5.2 Configuration

The configuration file is `/etc/squidclamav.conf`; one of the best points of SquidClamav is its ability to natively co-operate with [SquidGuard](#) (or any other redirector, specified in the `squidguard` directive), by calling it before any further processing of the HTTP request.

By default, SquidClamav will scan all downloaded files, but you can save system resources and improve performance by specifying a series of domains, users and clients (using the `whitelist`, `trustuser` and `trustclient` directives respectively) for which you want to skip both virus scanning and the call to the chained redirector. E.g.:

```
# Disable redirector and virus scanning for files from kernel-panic.it
whitelist      .*\.kernel-panic\.it

# Disable redirector and virus scanning for requests from a specific user
trustuser      administrator

# Disable redirector and virus scanning for requests from specific clients
trustclient    ^172\.16\.0\..*$
trustclient    ^mypc\.lan\.kernel-panic.it$
```

SquidClamav also allows you to disable virus scanning (but not the call to the chained redirector) based on regular expressions matching the URL string (abort directive) or the Content-Type of the response (abortcontent directive); both directives are case-insensitive. E.g.:

```
# Disable virus scanning for .png, .gif and .jpg images and documents from
# squid-cache.org
abort \.squid-cache\.org
abort .*\. (png|gif|jpg)$

# Disable virus scanning for images and flash videos
abortcontent ^image\/.*$
abortcontent ^video\/x-flv$
```

Below is a sample configuration file:

```
/etc/squidclamav.conf
```

```
# IP address and port of the Squid proxy
squid_ip      127.0.0.1
squid_port    3128

# Path to the log file
logfile       /var/log/squidclamav.log

# URL where to redirect a request when a virus is found. SquidClamav will
# append the original URL, virus name, source IP and user the to this URL
redirect      http://www.kernel-panic.it/viruswarn.php

# Disable virus scanning if the requested file hits squid cache
trust_cache   1

# Timeout when downloading files
timeout       60

# Set this to '1' for more verbose logging
debug         0

# Set this to '1' to force virus scan of URLs whose content-type can't be
# determined by libcurl
force         1

# Set this to '1' to show time statistics of URL processing
stat          0

# Don't follow more than 10 redirects
maxredirect   10

# Uncomment to disable virus scanning for files bigger than the specified
# size (in bytes)
# maxsize     2000000

# Uncomment to make cURL pretend to be Internet Explorer
#useragent    Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)

# IP address and port of the clamd daemon; you can specify up to 5
# (comma-separated) IPs for failover
clamd_ip      127.0.0.1
clamd_port    3310
# Uncomment if you're using the unix socket to communicate with clamd
#clamd_local  /tmp/clamd

# Check rules
abort         ^.*\. (ico|gif|png|jpg)$
```

```

abortcontent ^image\/*.*$
abort ^.*\.(css|xml|xsl|js|html|jsp)$
abortcontent ^text\/*.*$
abortcontent ^application\/*-javascript$
abortcontent ^video\/*mp4$
abortcontent ^video\/*-flv$
abort ^.*\.(pdf|swf)$
whitelist .*\.kernel-panic\.it

# Call another redirector (usually squidGuard) before the antivirus scanner
squidguard /usr/local/bin/squidGuard

```

Now we only have to modify the value of the `url_rewrite_program` parameter in Squid's configuration file:

```
/etc/squid/squid.conf
```

```
url_rewrite_program /usr/local/bin/squidclamav
```

and reload Squid.

```
# squid -k reconfigure
```

Note: to scan a file, SquidClamav needs to download it first; so make sure your [Squid ACLs](#) allow localhost to access the web:

```
/etc/squid/squid.conf
```

```
http_access allow localhost
```

You can check that everything is working fine by trying to download the [Eicar](#) anti-virus test file. In the log file, you should get something like:

```
/var/log/squidclamav.log
```

```

[...]
Fri May 15 19:26:49 2009 [29028] DEBUG received from Clamd: stream: Eicar-Test-
Signature FOUND
Fri May 15 19:26:49 2009 [29028] LOG Redirecting URL to: http://www.kernel-
panic.it/viruswarn.php?
url=http://www.eicar.org/download/eicar.com.txt&source=192.168.1.14/-&user=-
&virus=stream:+Eicar-Test-Signature+FOUND
Fri May 15 19:26:49 2009 [29028] DEBUG End reading clamd scan result.
Fri May 15 19:26:49 2009 [29028] DEBUG Virus found send redirection to Squid.

```

6. Ad Zapping with AdZapper

[AdZapper](#) is a “*redirector for squid that intercepts advertising (banners, popup windows, flash animations, etc), page counters and some web bugs (as found)*”. It will help users to get rid of those annoying popup windows, flash animations and malicious cookies and will help you save bandwidth and cache resources.

We will make use of three scripts:

- `squid_redirect`, which performs the actual ad zapping;
- `zapchain`, which chains multiple redirectors together (this is necessary because Squid accepts only one `redirector_program`);
- `wrapzap` which is a very simple wrapper script that sets environment variables useful to the redirector and then runs it.

6.1 Installation

The installation procedure is very simple. [Download](#) and extract the tarball, then copy the `squid_redirect`, `wrapzap` and `zapchain` scripts to `/usr/local/bin`, or wherever you prefer.

```
# tar -zxvf adzap-xxxxxxx.tar.gz
[...]
# cd adzap-xxxxxxx/scripts
# cp squid_redirect wrapzap zapchain /usr/local/bin/
```

The `zaps` directory contains the images that will replace the zapped ads: copy them to where the web server can find them. They're not really works of art, so feel free to customize them.

```
# scp -r ../zaps root@www.kernel-panic.it:/var/www/icons/
```

6.2 Configuration

AdZapper configuration takes place in the `wrapzap` script; below is a sample configuration script:

```
/usr/local/bin/wrapzap

#!/bin/sh

squidclamav=/usr/local/bin/squidclamav
zapper=/usr/local/bin/squid_redirect

# Setting ZAP_MODE to "CLEAR" will cause the zapper to use transparent images,
# thus completely hiding ads. This may, however, hide useful markup.
ZAP_MODE=

# Base URL of the directory containing the replacement images
ZAP_BASE=http://www.kernel-panic.it/icons/zaps
ZAP_BASE_SSL=https://www.kernel-panic.it/icons/zaps

# The following variables contain the path to extra pattern files.
# ZAP_PREMATCH patterns are consulted before the main pattern list. Use it to
# prevent overzapping by some erroneous patterns in the main pattern file.
ZAP_PREMATCH=

# ZAP_POSTMATCH patterns are consulted after the main pattern list. Use it to
# add extra patterns
ZAP_POSTMATCH=
```

```

# ZAP_MATCH patterns are consulted instead of the main pattern list. Use it to
# fully customize AdZapper
ZAP_MATCH=

# Should you use Apache2 instead of Squid, set this to "NULL"
ZAP_NO_CHANGE=

# Placeholder images names. "Clear" versions have "-clear" appended to the root
# portion of the file name; e.g. "ad.gif" becomes "ad-clear.gif".
STUBURL_AD=$ZAP_BASE/ad.gif
STUBURL_ADSSL=$ZAP_BASE_SSL/ad.gif
STUBURL_ADBG=$ZAP_BASE/adbg.gif
STUBURL_ADJS=$ZAP_BASE/no-op.js
STUBURL_ADJSTEXT=
STUBURL_ADHTML=$ZAP_BASE/no-op.html
STUBURL_ADHTMLTEXT=
STUBURL_ADMP3=$ZAP_BASE/ad.mp3
STUBURL_ADPOPOP=$ZAP_BASE/closepopup.html
STUBURL_ADSWF=$ZAP_BASE/ad.swf
STUBURL_COUNTER=$ZAP_BASE/counter.gif
STUBURL_COUNTERJS=$ZAP_BASE/no-op-counter.js
STUBURL_COUNTERHTML=$ZAP_BASE/no-op-counter.html
STUBURL_WEBDEBUG=$ZAP_BASE/webbug.gif
STUBURL_WEBDEBUGJS=$ZAP_BASE/webbug.js
STUBURL_WEBBUGHTML=$ZAP_BASE/webbug.html

# Set this to "1" to use the rewrite facility to get the printer-friendly
# version of some pages
STUBURL_PRINT=

export ZAP_MODE ZAP_BASE ZAP_BASE_SSL ZAP_PREMATCH ZAP_POSTMATCH ZAP_MATCH
ZAP_NO_CHANGE
export STUBURL_AD STUBURL_ADSSL STUBURL_ADJS STUBURL_ADHTML STUBURL_ADMP3 \
        STUBURL_ADPOPOP STUBURL_ADSWF STUBURL_COUNTER STUBURL_COUNTERJS \
        STUBURL_COUNTERHTML STUBURL_WEBDEBUG STUBURL_WEBDEBUGJS STUBURL_WEBBUGHTML \
        STUBURL_PRINT STUBURL_ADHTMLTEXT STUBURL_ADJSTEXT

# Exec the real zapper (chained with SquidClamav)
exec /usr/local/bin/zapchain "$zapper" "$squidclamav"

```

Now we only have to update the `url_rewrite_program` in Squid's configuration file:

```
/etc/squid/squid.conf
```

```
redirect_program /usr/local/bin/wrapzap
```

and reload Squid.

```
# squid -k reconfigure
```

Now ads should magically disappear from web sites!

7. Appendix

7.1 Tunneling through Squid

So you have finally configured your proxy server, allowing only requests to a few standard ports, blocking blacklisted sites, ads and viruses. The HTTP CONNECT method is restricted to the standard HTTPS port. Your LAN firewalls rules are very strict and block everything but requests to port 3128 of the proxy. Therefore, you feel pretty confident that users won't be able to do anything on the Internet you didn't explicitly allow.

But Squid is an ugly beast, and if you don't pay very close attention to its configuration (and log files), your users could end up getting around most of your blocking rules. Let's have a look at a practical example.

[Stunnel](#) is a program that allows you to encrypt arbitrary TCP connections inside SSL. It is mainly used to secure non-SSL aware daemons and protocols (like POP, IMAP, LDAP, etc) by having Stunnel provide the encryption, requiring no changes to the daemon's code.

Basically, Stunnel establishes an encrypted and persistent connection between two separate machines. One machine acts as the server and forwards any connection Stunnel receives to a user-defined port. The other machine acts as the client, binding to an arbitrary port and forwarding any connection it receives on that port to the server machine.

We will use Stunnel and Squid to bypass firewall rules and [ssh\(1\)](#) to a remote server (e.g. your home computer) from a local computer in the corporate LAN. The OpenBSD ports and packages archives include a few similar tools for tunneling network traffic through proxy servers, such as:

- [Corkscrew](#), a tool for tunneling [ssh\(1\)](#) through HTTP proxies;
- [gotthard](#), a daemon which tunnels [ssh\(1\)](#) sessions through an HTTPS proxy;
- [httptunnel](#), which creates a bidirectional virtual data connection tunnelled in HTTP requests.

However, Stunnel is probably the most versatile and comprehensive tunneling solution, since it can forward any type of network traffic (not only [ssh\(1\)](#)) and provides an additional SSL cryptography layer, thus protecting clear text protocols such as [telnet\(1\)](#) or [ftp\(1\)](#).

7.1.1 Server-side configuration

The remote computer will necessarily have to act as the server. Install [stunnel](#) from the packages and create the SSL certificate:

```
remote# pkg_add stunnel-x.xx.tgz
[...]
remote# openssl req -x509 -newkey rsa:2048 -keyout /etc/ssl/private/stunnel.key \
> -out /etc/ssl/stunnel.crt -days 365 -nodes
[...]
remote# chmod 600 /etc/ssl/private/stunnel.key
```

Then configure Stunnel to bind to port 443 (HTTPS) and forward incoming connections to port 22 (ssh). The configuration file is `/etc/stunnel/stunnel.conf`:

```
/etc/stunnel/stunnel.conf
```

```
cert = /etc/ssl/stunnel.crt
key = /etc/ssl/private/stunnel.key

chroot = /var/stunnel/
setuid = _stunnel
```

```

setgid = _stunnel
pid = /var/run/stunnel.pid
sslVersion = SSLv3

socket = l:TCP_NODELAY=1
socket = r:TCP_NODELAY=1

[https]
accept = 443
connect = 22
TIMEOUTclose = 0

```

Now we can start it and go to work to have some fun with our tunnel:

```
remote# /usr/local/sbin/stunnel
```

7.1.2 Client-side configuration

So now we come to the local computer, which will act as the client. The SSL tunnel needs to go through Squid to get around the firewall rules but, by default, Stunnel doesn't support web proxies. Fortunately, a few patches are available that add SSL-proxy support to Stunnel. The [most recent available](#) applies to Stunnel version 4.05, so I suggest that you [download](#) and install this version on the client machine.

```

local$ tar -zxvf stunnel-4.05.tar.gz
[...]
local$ patch -p0 < connect-proxy.mwald.patch
[...]
local$ cd stunnel-4.05
local$ ./configure
[...]
local$ ln -s /usr/sbin/openssl /usr/bin/openssl
local$ make
[...]
local$ su
Password:
local# make install
[...]
local#

```

The patch introduces two additional configuration parameters: `httpsproxy_dest` (name or address of the Stunnel server) and `httpsproxy_auth` (proxy authentication credentials). We will configure the client to accept connections on an arbitrary port (e.g. 1443) and forward them to port 443 of the remote Stunnel server (which, in turn, will forward them to port 22). In other words, when you will connect to port 1443 on the local computer, you will actually get connected to port 22 on the remote computer.

The client configuration file looks like:

```
/etc/stunnel/stunnel.conf
```

```

chroot = /var/stunnel
setuid = _stunnel
setgid = _stunnel
pid = /var/run/stunnel.pid

client = yes

[https]
accept = 1443
connect = web-proxy:3128
httpsproxy_dest = stunnel-server:443

```

```
httpsproxy_auth = username:password
```

Ok, everything is ready, let's give it a try:

```
local# /usr/local/sbin/stunnel
local# ssh localhost -p 1443
root@localhost's password:
remote#
```

As you can see, despite firewall rules and Squid ACLs, we have successfully connected to the remote computer. Once the tunnel is up, you could even do the opposite and connect from the remote server to the local client by simply opening a reverse ssh from the local client:

```
local# ssh -NR 2443:localhost:22 -p 1443
```

This way, every connection received by the remote server on port 2443 will be forwarded to port 22 of the local client:

```
remote# ssh localhost -p 2443
root@localhost's password:
local#
```

You could even allow X11 forwarding on the remote server and have your whole remote graphical environment available on the local machine (for instance to surf the web with no proxy filters).

Anyway, this paragraph only meant to point out how much careful Squid configuration must be. Usually, however, the stricter your corporate policy, the more determined your users will be to evade it.

By the way, using whitelists is probably the best solution to prevent tunneling, but, if they are too restrictive, get ready to get your car keyed by a crowd of angry users!

7.2 References

- [OpenBSD](#), the secure by default operating system
- [Squid](#), a full-featured Web proxy cache designed to run on Unix systems
- [Squidguard](#), an ultrafast and free filter, redirector and access controller for Squid
- [ClamAV](#), a GPL anti-virus toolkit for UNIX
- [SquidClamav](#), a Clamav Antivirus Redirector for Squid
- [AdZapper](#), a redirector for squid that intercepts advertising, page counters and some web bugs
- [DansGuardian](#), *true* web content filtering for all
- [Stunnel](#), the universal SSL wrapper
- [HTTP Connect-style proxy patch for Stunnel](#)
- [Corkscrew](#), a tool for tunneling [ssh\(1\)](#) through HTTP proxies
- [gotthard](#), a daemon which tunnels [ssh\(1\)](#) sessions through an HTTPS proxy
- [httptunnel](#), a tool for creating a bidirectional virtual data connection tunnelled in HTTP requests

7.3 Bibliography

- [The Squid Documentation Project](#)
- [Squid Frequently Asked Questions](#)
- [Squid Wiki](#)
- [Squid configuration manual](#)
- [Squid-Book oltre le FAQ \(Italian only\)](#)
- [Configuring squidGuard](#)
- [Meeting the Challenges of Web Content Filtering](#)

