Transparent Proxy with contentfilter

Introduction

This article describes the setup of a http proxy chained to a content filter (think “parental control” for instance).

There are many cases, where is not desirable to grant unlimited Internet access to certain groups of people. The most obvious group are children, either at home, in schools or public libraries, whom you want to protect from hitting upon too explicit imagery or language. Or perhaps you just want to block certain sites from them.
Other uses for a content-filtering web proxy would be companies that want to limit the accessibility of the Internet facility they allow their employees. Content filtering does not stop at blocking undesirable content - blocking viruses in downloaded materials and malicious HTML code is another form of filtering incoming web traffic.
A web proxy is what you need in this case. The proxy server intercepts all browser requests for web pages, and in co-operation with one or more filtering programs, decides whether the browser will or won't be able to retrieve the full content it was requesting.

The problem (if you want to call it that) is that the end user's web browser has to be re-configured to use a proxy server. Normally, a browser's request for a web page is sent directly to the remote Internet webserver. Letting a proxy server do all the content filtering means that the browser will have to know about the proxy and hand over every browser request to it. If you have a large network, this means that you will have to find a way of configuring all your desktops to use the proxy... and make sure the end users will *keep using the proxy*!
It would be trivial for users to just disable the proxy configuration and circumvent your content filter... unless you block all outgoing internet traffic except for the traffic originating from the proxy server.

Ideally, you would want a solution where such a filtering proxy server is installed on the Internet Gateway Server, and the desktop computers would not have to be be reconfigured to make use of the proxy. This is called *transparent proxying* - the proxy will intercept the browser requests without the end users knowing their traffic is monitored - until they hit a censored page of course!

I will show you how you can use tinyproxy and dansguardian in combination with a few iptables firewall rules, to accomplish such a transparent proxy service. I chose tinyproxy because it is small and very fast in comparison with the more widely used Squid proxy. Tinyproxy is just fine for small to medium-sized networks. For larger networks, you can replace tinyproxy with Squid without much effort and it will still work (if you configure Squid right). But I will concentrate on tinyproxy for now.

Tinyproxy and Squid are licenced under the GPL. Dansguardian is licenced under the GPL, with the addition that it is free for non-commercial use.

This article focuses on the configuration of a transparent proxy on a gateway/router for **small networks**. Another scenario is that of the **family computer** with a single network interface, running Linux, where you want to restrict the children in their Internet browsing while still being able to have unrestricted Internet access for your own account (assuming *you* are the parent) or as the root user.

I have a Wiki page that points out the different steps you need to take compared to this very page here.

When using this proxy/contentfilter, it will not be possible for the content filter to examine *HTTPS* requests. This is of course due to the nature of the encryption used - if it *were* possible for the content filter to examine the content of secure HTTPS connections, then this would pose a serious threat to all secure communication on the Internet. This would be called the “man in the middle attack”.
The tinyproxy by itself can proxy the HTTPS traffic because *it* does not need to inspect the content of the HTTPS traffic, it just passes the received data on to the client browser. This is the reason why in the rest of the article, there will be a few examples of redirecting HTTPS traffic (tcp port 443); it is only for the benefit of people who use this article to just setup a proxy without filtering.

How it works

The usual flow of traffic in a network without proxies of any kind is like this:

```
browser -> (gateway) -> (internet) -> webserver
```

The client browser's request for external server pages is routed through the Internet Gateway which is also the internal network's default gateway.

With a proxy/contentfilter installed on the gateway, and without transparency, the browser must be configured to talk directly to the proxy. The flow would change to

```
browser -> (contentfilter -> proxy on gateway) -> internet -> webserver
```

The browser talks to the address/port it configured as the “proxy server”, this being `192.168.2.1:8080`. In fact, it is the content filter listening at that port. The content filter (dansguardian in our case) will decide whether or not to deliver the content back to the browser. For each HTTP request, the content filter passes the request to the proxy server (tinyproxy in our case) and leaves the process of retrieving the web content to the proxy. Fetched pages or other content (think of file downloads) are passed back to the content filter which inspects what it got handed over by the proxy. HTML pages are scanned for undesired content and (optionally) harmful code; downloaded files are (optionally) scanned for viruses; and if the requested URL matches with an entry in a blacklist, the proxy will not even have to fetch the page - because the content filter will display a “URL blocked” warning page instead.

Now, when the gateway is configured for transparent proxying, the browser does not (have to) know this. The flow is like this:

```
browser -> (gateway eth1)....(gateway eth0) -> internet -> webserver
           (iptables)                    ^
             |                           |
             ->(contentfilter -> proxy) _|
```

The browser request is again routed through the default gateway as in the first picture, only now we have an `iptables` rule in place that detects the traffic targeted at an external web server (ports 80 or 443). The iptables rule causes this traffic to be *re-directed* to the port where our content filter is listening (`192.168.2.1:8080`)!
The browser is unaware of this “hi-jacking” process. Browser requests will be examined by the contentfilter and URLs that appear in a blacklist will trigger an immediate *access denied* page to be returned to the browser. Other page requests are retrieved by the proxy and then examined and scanned, just like was shown in the second picture.

I touched on the issue of scanning for viruses and malicious HTML code - dansguardian can not do this by itself, but if compiled with the appropriate support (see below) and configured to actually use it, dansguardian will contact an available ClamAV virus scanning daemon and let it do the scanning.

Network layout

For the sake of simplicity, I will assume the proxy server and content filter will be installed on the server that is also acting as the Internet Gateway. This server has two network interfaces, one connecting to your ADSL/Cable router and the other is connected to your internal network.
This also means that this server's IP address is configured as the default gateway on the computers in your network. If the proxy server is going to be installed on another server than the Internet Gateway, you will have to do some additional work to redirect traffic from the Gateway to the proxy server and back out through the Gateway to the Internet. This is left as an excercise to the reader 

The TCPIP configuration of our network will be as follows:

Table 1. Server:

Interface	Type	IP address	Netmask	Default gateway
eth0	dynamic	10.111.111.1	255.255.255.128	10.111.111.254
eth1	static	192.168.2.1	255.255.255.0	

Table 2. Internal Network:

Network	Netmask	Default Gateway
192.168.2.0	255.255.255.0	192.168.2.1

Table 3. Listen ports:

Program	IP address	portnumber
dansguardian	192.168.2.1	8080
tinyproxy	127.0.0.1	3128

The server's eth0 interface address is dynamically assigned by your ISP's DHCP server - I picked a random address. This address will re-appear in some of the configuration files below. If your own setup uses other IP addresses, just change where appropriate and/or indicated. The default gateway

which is mentioned in *Table 1* will be assigned by the ISP's DHCP server, so you won't have to configure that one explicitly.
If you run a firewall on the server, you will have to make sure the DHCP handshake is not blocked. More about that later on.

The server's eth1 interface is connected to your internal network. Ideally you would be running your own DHCP server on this interface, so that the computers in your network don't have to be configured with IP addresses manually. How to setup a DHCP server is something for a future article perhaps.

The network configuration file `/etc/rc.d/rc.inet21.conf` for your server will look like this:

```
# Config information for eth0:
IPADDR[0]=""
NETMASK[0]=""
USE_DHCP[0]="yes"
DHCP_HOSTNAME[0]=""

# Config information for eth1:
IPADDR[1]="192.168.2.1"
NETMASK[1]="255.255.255.0"
USE_DHCP[1]=""
DHCP_HOSTNAME[1]=""

# Default gateway IP address:
GATEWAY=""
```

Getting and building the software

Building tinyproxy

The homepage for tinyproxy has download links for “stable” and “devel” versions. Since the “devel” release 1.7.0 is alreay quite old, I decided to just use that. From the mailing list it also became clear that the current maintainer is looking for someone to take over the project.
Tinyproxy must explicitly be build for transparent proxy support. If you want to compile and install the software manually, this is what you would do:

```
tar -zxvf tinyproxy-1.7.0.tar.gz
cd tinyproxy-1.7.0
./configure --prefix=/usr \
            --localstatedir=/var \
            --sysconfdir=/etc \
            --enable-xtinyproxy \
            --enable-filter \
            --enable-upstream \
            --enable-reverse \
            --enable-transparent-proxy \
            --program-prefix="" \
            --program-suffix=""
make
```

```
make install
```

If you rather install a Slackware package or want to use a SlackBuild script to create a Slackware package, you'll find all you need in my repository

Building dansguardian

The dansguardian software is actively maintained. You will need the basic software package you can download from the dansguardian homepage and the default configuration will already be sufficient for a lot of people. If you want more extensive URL blacklists or badword lists you can look at the website. Some extensions you'll find have to be paid for however.
Although the most current release is in the *ALPHA* download section, it's actually quite stable. I used that for my install. For the manually compiling people:

```
tar -zxvf dansguardian-2.9.7.0.tar.gz
cd dansguardian-2.9.7.0
./configure --prefix=/usr \
            --localstatedir=/var \
            --sysconfdir=/etc \
            --enable-pcre \
            --enable-clamd \
            --enable-email \
            --with-proxyuser=nobody \
            --with-proxygroup=nobody
make
make install
```

I have a SlackBuild and a Slackware package for dansguardian in my repository which you can use as well. The advantage being that I added a start script and a logrotate script to the package. If you want those without building from my SlackBuild script, I added them in the last section.
I configured dansguardian to run as user *nobody* - because that is an existing account without provileges, and Apache uses it too. If you want another account change the `./configure` step, and create the account you want it to use in case the account does not yet exist. We will configure tinyproxy to run as user *nobody* as well, but in tinyproxy's case, we don't have to define that at compile-time. Tinyproxy has the effective user as a parameter in it's configuration file (see below).

Configuration

That was it! Now, it is time to start configuring our proxying service.

tinyproxy config

The content of the tinyproxy configuration file `/etc/tinyproxy/tinyproxy.conf` (stripped of comments and empty lines) for our example setup can be found in the last section.
I entered the domain name for my internal lan *my.net* in this configuration file. If yours is different, please change accordingly.

To show you where this differs from the tinyproxy defaults, here is a diff from the original file:

```
diff /etc/tinyproxy/tinyproxy.conf-dist /etc/tinyproxy/tinyproxy.conf
20c20
< Port 8888
---
> Port 3128
27c27
< #Listen 192.168.0.1
---
> Listen 127.0.0.1
34c34
< #Bind 192.168.0.1
---
> Bind 10.111.111.1
112c112
< #XTinyproxy mydomain.com
---
> XTinyproxy my.net
192c192
< Allow 192.168.1.0/25
---
> Allow 192.168.2.0/24
```

The important lines here are as follows:

```
Port 3128
Listen 127.0.0.1
Bind 10.111.111.1
Allow 127.0.0.1
Allow 192.168.2.0/24
```

These achieve the following:

- make tinyproxy listen on `127.0.0.1:3128` where dansguardian will contact it
- bind to the external interface (IP address `10.111.111.1`) which makes all traffic direction Internet originate from the external interface (this is a required line for a host with multiple network interfaces)
- allow the localhost (IP address 127.0.0.1, for dansguardian) as well as all the computers in your internal LAN (IP address range 192.168.2.0-192.168.2.255) access -implicitly denying access attempts from any other IP address but those.

dansguardian config

You will find the content of the dansguardian configuration file `/etc/dansguardian/dansguardian.conf` (stripped of comments and empty lines) for our example setup in the last section. The difference with the originally distributed file is quite small:

```
diff /etc/dansguardian/dansguardian.conf.new
/etc/dansguardian/dansguardian.conf
```

```
48a50
> anonymizelogs = off
74c76
< filterip =
---
> filterip = 192.168.2.1
97c99
< accessdeniedaddress =
'http://YOURSERVER.YOURDOMAIN/cgi-bin/dansguardian.pl'
---
> accessdeniedaddress = 'http://192.168.2.1/cgi-bin/dansguardian.pl'
```

The important lines in this file are:

```
filterip = 192.168.2.1
filterport = 8080
proxyip = 127.0.0.1
proxyport = 3128
```

They show that dansguardian

- will listen at the *filterip/filterport* address, i.e. `192.168.2.1:8080`. Sound familiar? This is the Proxy URL! Dansguardian is the primary entry point for the browsers' http requests.
- will look for a compatible proxy at IP address:port `127.0.0.1:3128`. This matches exactly with how we configured tinyproxy.

The line

```
accessdeniedaddress = 'http://192.168.2.1/cgi-bin/dansguardian.pl'
```

does not really matter, because in dansguardian's (and our) default configuration, the *access denied* web page is generated from a language-dependent template page. That is why you define `language = 'ukenglish'` - if you want dansguardian to show it's messages in another language, look in directory `/usr/share/dansguardian/languages/` for the available languages.

Of course, there is a lot of fine-tuning possibilities in this configuration file, as well as many others in the `/etc/dansguardian` directory tree. But in it's default setup dansguardian already does some impressive (perhaps *aggressive* is the better word) filtering.

The iptables firewall

For a transparent proxy setup, you will need to be running a *NAT* or *masquerading* router/gateway. Your Slackware Linux server comes with the iptables tool, which is commonly used to setup such a NAT router. A NAT router has (at least) two network interfaces, one connected to the “outside world” - to your ISP's DSL modem for instance - and the other network interface will be connected to your internal LAN. The NAT router will make your computers in the internal network invisible and inaccessible to anyone outside. The NAT router will also allow all computers in the internal network to access the outside world (the Internet) using it's single external interface.
The commodity hardware Cable/DSL routers you can buy in any computer shop work exactly the

same; it's just that with a real Linux server you have so much more power at hand.

So, back to our server... You will either have to setup an iptables firewall script on your Internet Gateway, or modifiy an existing iptables firewall ruleset.
The Slackware boot scripts expect your firewall script to be named

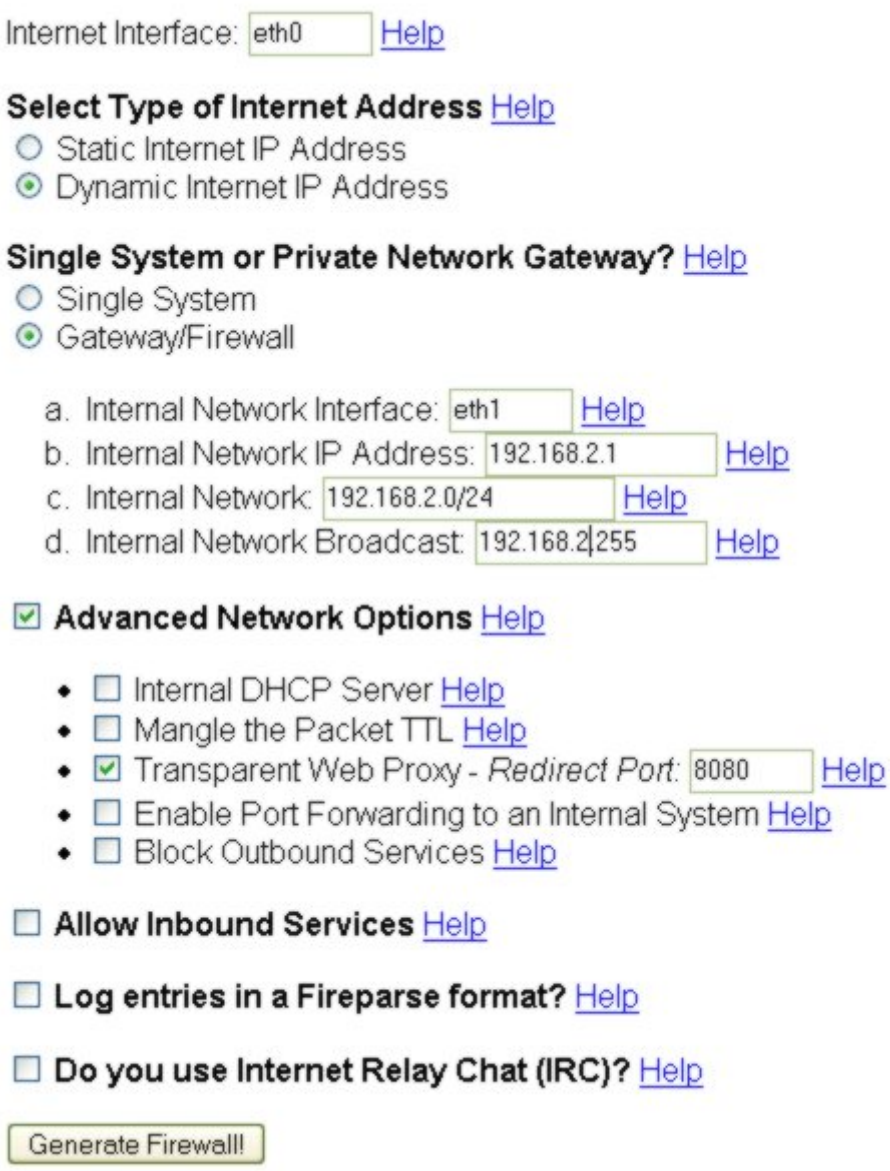
```
/etc/rc.d/rc.firewall
```

and this script should be executable:

```
chmod +x /etc/rc.d/rc.firewall
```

. The contents of the script can be anything you like, as long as it installs a firewall ruleset of course 

- If you are in the process of setting up a router/gateway/proxy and want an comfortable way of creating a solid iptables firewall, use the Easy Firewall Generator which I installed in with some small modifications to the original script. The modifications make the resulting generated script work with Slackware.
  You can copy the script generated by the aforementioned “Easy Firewall Generator” and paste the content into the `rc.firewall` file. This is how your settings should look right before the actual script is generated:



The generated script enables transparent proxying of all browser requests targeted at external servers on port 80 (HTTP traffic). If you also want to re-direct the HTTPS traffic (port 443) you will need to open the `rc.firewall` script, scroll all the way down, and enable the line that mentions the port 443:

```
# Redirect HTTPS for a transparent proxy - commented by default
$IPT -t nat -A PREROUTING -p tcp --destination-port 443 \
     -j REDIRECT --to-ports 8080
```

A few lines further up in the same script, you will also see how to make exceptions for specific IP addresses to circumvent the proxy (perhaps the Systems Administrator will need unrestricted Internet access?):

```
# This is a sample that will exempt a specific host from the
transparent proxy
#$IPT -t nat -A PREROUTING -p tcp -s 192.168.1.50 --destination-port 80
\
#     -j RETURN
#$IPT -t nat -A PREROUTING -p tcp -s 192.168.1.50 --destination-port
443 \
```

```
#     -j RETURN
```

Just remove the `#` from the start of these lines and replace `192.168.1.50` with the sysadmin's IP address.

- For those who already have an iptables firewall script in place and want to know what they have to add in order to enable the transparent proxying; these are the important lines. Add them somewhere in your script.

```
# Redirect HTTP for a transparent proxy
/usr/bin/iptables -A PREROUTING -t nat -p tcp --destination-port 80 \
   -j REDIRECT --to-ports 8080
# Redirect HTTPS for a transparent proxy - commented by default
/usr/bin/iptables -A PREROUTING -t nat -p tcp --destination-port 443 \
   -j REDIRECT --to-ports 8080
```

Starting the programs

If you (built and) installed my Slackware package for dansguardian, the rc script is installed non-executable by default. In order to run dansguardian on boot (as shown below) you will have to make the script executable by running

```
chmod +x /etc/rc.d/rc.dansguardian
```

If you configured your firewall rules in the file `/etc/rc.d/rc.firewall`, then this script will be detected by Slackware and automatically started with the `start` parameter on boot. This happens in the the Slackware init script `/etc/rc.d/rc.inet2` to be precise, like this:

```
if [ -x /etc/rc.d/rc.firewall ]; then
  /etc/rc.d/rc.firewall start
fi
```

so we don't have to worry about that. The important bit is the order in which tinyproxy and dansguardian are started: if dansguardian does not find a proxy service listening at the configured address, it will refuse to start. So, we add these lines to the file `/etc/rc.d/rc.local`:

```
if [ -x /usr/sbin/tinyproxy ]; then
  /usr/sbin/tinyproxy > /dev/null 2>&1
fi

# Start dansguardian
if [ -x /etc/rc.d/rc.dansguardian ]; then
  /etc/rc.d/rc.dansguardian start
fi
```

Both programs log their actions; to respectively `/var/log/tinyproxy.log` and `/var/log/dansguardian/access.log`. If their logging is a bit too verbose to your taste (the default is to log a *lot*) you can turn the log levels down in the configuration files.

That is all there is to it! Now, test your rig by booting a client computer in your LAN and trying out a couple of URLs. I will leave it to your own imagination as to what URLs will be considered *naughty*.

To stop these programs if you need to, you run

```
/etc/rc.d/rc.dansguardian stop
killall -TERM tinyproxy
```

Adding virus scanning

UNDER CONSTRUCTION


Pitfalls

There might be cases where you don't want transparent proxying. For instance, some applications will not correctly connect through a transparent proxy. The (client side) user agent does not know it passes a proxy, so it possibly will not send correct HTTP headers to the remote server. Most modern browsers are standards-compliant however and will work fine. If your users need Internet Explorer, it must be newer than 5.5_SP1. For those cases where transparent proxying is impossible, you must configure your browsers explicitly to use the proxy. How to do this in a semi-centralized way is described in the next section. Web browsers that are specifically configured to use a proxyhave no problems connecting to external servers, because they “know” they use a proxy and add cache-aware HTTP headers to each request. Because of that, the remote server knows the client is behind a proxy and adjusts it's behaviour.

Manual proxy configuration

A handy way of browser configuration is the *Proxy Auto-Configuration* or PAC standard. On one of your internal webservers you install a “pac” file with proxy parameters, and then you instruct all your browsers to go fetch that “pac” file and interpret it's directives. The advantages to just entering the proxy server's IP address and port number are, that you can define a much more fine-grained configuration in the “pac” file, and you can change it's contents without having to re-do your manual browser configuration. The “pac” file will be downloaded and interpreted *every* time a browser starts.

You can read more about PAC on the Netscape web pages (yes, Netscape ). This is also an excellent page.

To give an example, create a file on your Gateway server's *DocumentRoot* and call it `proxy.pac`. This will make it available under the URL “http://192.168.2.1/proxy.pac”. Let the contents of the `proxy.pac` file be like this:

```
function FindProxyForURL(url, host)
{
  //Traffic to the localhost should go directly;
```

```
  //If they have only specified a hostname, go directly.
  if (isInNet(host, "127.0.0.0", "255.0.0.0") || isPlainHostName(host)) {
    return "DIRECT";
  //So the error message "no such host" will appear through the
  //normal browser dialog box - less support queries :)
  if (!isResolvable(host))
    return "DIRECT";
  // Don't use a proxy inside our own LAN
  else if (isInNet(host, "192.168.2.0", "255.255.0.0")) {
    return "DIRECT";
  //We only cache https and http
  else if (url.substring(0, 5) == "http:" ||
    url.substring(0, 4) == "https:")
    return "PROXY 192.168.2.1:8080; DIRECT";
  // Catch-all
  else
    return "DIRECT";
}
```

This essentially tells the browser to use `192.168.2.1:8080` as the proxy server address, and if that address does not responds as a proxy (because it is down for instance) use the `DIRECT` instruction, i.e. circumvent the proxy and try to make a direct connection.

If your webserver is Apache, and your browsers spits out an error on the "http://192.168.2.1/proxy.pac" setting, you might have to add the correct MIME type to your Apache server. These lines should go into `/etc/apache/httpd.conf` for instance:

```
#
# MIME type for proxy autoconfiguration:
#
AddType application/x-ns-proxy-autoconfig .pac
```

after which the webserver should be restarted.

Example configuration files

`/etc/tinyproxy/tinyproxy.conf`:

```
User nobody
Group nogroup
Port 3128
Listen 127.0.0.1
Bind 10.111.111.1
Timeout 600
DefaultErrorFile "/usr/share/tinyproxy/default.html"
StatFile "/usr/share/tinyproxy/stats.html"
Logfile "/var/log/tinyproxy.log"
LogLevel Info
PidFile "/var/run/tinyproxy.pid"
```

```
XTinyproxy my.net
MaxClients 100
MinSpareServers 5
MaxSpareServers 20
StartServers 10
MaxRequestsPerChild 0
Allow 127.0.0.1
Allow 192.168.2.0/24
ViaProxyName "tinyproxy"
ConnectPort 443
ConnectPort 563
```

/etc/dansguardian/dansguardian.conf:

```
reportinglevel = 3
languagedir = '/usr/share/dansguardian/languages'
language = 'ukenglish'
loglevel = 2
logexceptionhits = on
logfileformat = 1
anonymizelogs = off
filterip = 192.168.2.1
filterport = 8080
proxyip = 127.0.0.1
proxyport = 3128
accessdeniedaddress = 'http://192.168.2.1/cgi-bin/dansguardian.pl'
nonstandarddelimiter = on
usecustombannedimage = 1
custombannedimagefile = '/usr/share/dansguardian/transparent1x1.gif'
filtergroups = 1
filtergroupslist = '/etc/dansguardian/lists/filtergroupslist'
bannediplist = '/etc/dansguardian/lists/bannediplist'
exceptioniplist = '/etc/dansguardian/lists/exceptioniplist'
showweightedfound = on
weightedphrasemode = 2
urlcachenumber = 1000
urlcacheage = 900
scancleancache = on
phrasefiltermode = 2
preservecase = 0
hexdecodecontent = 0
forcequicksearch = 0
reverseaddresslookups = off
reverseclientiplookups = off
logclienthostnames = off
createlistcachefiles = on
maxuploadsize = -1
maxcontentfiltersize = 256
maxcontentramcachescansize = 2000
maxcontentfilecachescansize = 20000
filecachedir = '/tmp'
```

```
deletedownloadedtempfiles = on
initialtrickledelay = 20
trickledelay = 10
downloadmanager = '/etc/dansguardian/downloadmanagers/fancy.conf'
downloadmanager = '/etc/dansguardian/downloadmanagers/default.conf'
contentscannertimeout = 60
contentscanexceptions = off
recheckreplacedurls = off
forwardedfor = off
usexforwardedfor = off
logconnectionhandlingerrors = on
logchildprocesshandling = off
maxchildren = 120
minchildren = 8
minsparechildren = 4
preforkchildren = 6
maxsparechildren = 32
maxagechildren = 500
maxips = 0
ipcfilename = '/tmp/.dguardianipc'
urlipcfilename = '/tmp/.dguardianurlipc'
ipipcfilename = '/tmp/.dguardianipipc'
nodaemon = off
nologger = off
logadblocks = off
softrestart = off
mailer = '/usr/sbin/sendmail -t'
```

/etc/rc.d/rc.dansguardian:

```
#!/bin/sh
#
# Startup script for dansguardian
#
# processname: dansguardian
# pidfile: /var/run/dansguardian.pid
# config: /etc/dansguardian/dansguardian.conf

# File includes changes by Thomas Jarosch
function wait_for_pid()
{
    local PID=$1
    local RET=0
    if [ $PID -eq 0 ] ; then
        return $RET
    fi
    # give 60 secs then KILL
    local COUNTDOWN=60

    while [ -d /proc/${PID} ] && [ $COUNTDOWN -gt 0 ] ; do
        sleep 1
```

```
        COUNTDOWN=$[$COUNTDOWN-1]
    done

    if [ -d /proc/${PID} ]; then
        COMMAND=`ps h -o command ${PID}`
        logger "dansguardian: timeout waiting for PID ${PID}: ${COMMAND};
sending SIGKILL"
        kill -KILL $PID >/dev/null 2>&1
        RET=1
    fi
    return $RET
}

# See how we were called.

case "$1" in
start)
        if [ -f /usr/sbin/dansguardian ] &&
           [ -f /etc/dansguardian/dansguardian.conf ]; then
                echo -n "Starting dansguardian: "
                if /usr/sbin/dansguardian 2> /dev/null; then
                        echo " OK"
                else
                        echo " FAILED"
                fi
        fi
        ;;
stop)
        echo -n "Shutting down dansguardian: "
        WAITPID=0
        if [ -f /var/run/dansguardian.pid ] ; then
            WAITPID=`cat /var/run/dansguardian.pid`
        fi
        if /usr/sbin/dansguardian -q 2> /dev/null; then
                if wait_for_pid $WAITPID ; then
                    echo " OK"
                else
                    echo " FAILED"
                fi
                /bin/rm -f /var/run/dansguardian.pid
                /bin/rm -f /tmp/.dguardianipc
        else
                echo " FAILED"
        fi
        ;;
restart)
        $0 stop
        $0 start
        ;;
status)
        if [ -x /usr/sbin/dansguardian ]; then
```

```
                /usr/sbin/dansguardian -s
    else
        echo "Dansguardian is not installed!"
        fi
        ;;
*)
        echo "Usage: $0 {start|stop|restart|status}" >&2
        ;;
esac

exit 0
```

/etc/logrotate.d/dansguardian:

```
/var/log/dansguardian/access.log {
  rotate 4
  weekly
  sharedscripts
  prerotate
    /usr/sbin/dansguardian -q > /dev/null
    sleep 5
  endscript

  postrotate
    /usr/sbin/dansguardian > /dev/null
  endscript
}
```