

ProxyScan

Viren scannen am Web-Proxy

Wolfgang Barth¹

April 2004

¹wob@swobspace.de

Zusammenfassung

Dieses Dokument beschreibt Möglichkeiten für den Einsatz von Virenscannern beim Websurfen unter Nutzung von OpenSource-Software. Im Normalfall erfolgt dies in Kombination mit einem Proxy.

Der Einsatz von Virenscannern für Email ist nicht Gegenstand dieses Dokumentes.

Alle in diesem Dokument enthaltenen Programme, Darstellungen und Informationen wurden sorgfältig zusammengestellt geprüft. Trotzdem sind Fehler nie ganz auszuschließen. Der Autor übernimmt daher keinerlei Garantie für eine fehlerfreie Funktionalität, die sich aus der Konfiguration nach den hier gemachten Angaben ergibt.

Alle Warennamen, Handelsbezeichnungen und Gebrauchsnamen sind eingetragene Warenzeichen der jeweiligen Eigentümer. Der Begriff Windows wird generisch benutzt für die Betriebssysteme von Microsoft. Microsoft, Windows, Windows 2000, Windows XP und andere Namen sind Marken und/oder eingetragene Warenzeichen von Microsoft.

Eine Wiedergabe von Warennamen, Handelsbezeichnungen und Gebrauchsnamen hier auch ohne besondere Kennzeichnung berechtigt nicht zur freien Verwendung derselben.

©Alle Rechte bei Wolfgang Barth. Das Dokument kann für den eigenen, auch firmeninternen Gebrauch (etwa im Intranet) weitergegeben werden, wenn es unverändert bleibt und ein vollständiger, eindeutiger Quellenhinweis auf die Urheberschaft hinweist.

Inhaltsverzeichnis

1	Einführung	3
1.1	Grundprinzip	3
1.2	Virensan und Browsergeduld	5
1.3	Übersicht über die unterschiedlichen Lösungen	5
1.3.1	squid-vscan	5
1.3.2	Internet Content Adaption Protocol ICAP	6
1.3.3	Viralator	7
1.3.4	Apache::ProxyScan	8
1.3.5	Apache2 und mod_clamav	8
1.3.6	Apache2 und mod_vscan	9
2	Apache::ProxyScan	11
2.1	Vorbereitungen	11
2.2	Squid-Konfiguration	12
2.3	Apache-Konfiguration	12
3	Apache2 Modul: mod_clamav	15
3.1	Vorbereitungen	15
3.2	Squid-Konfiguration	16
3.3	Apache-Konfiguration	17
A	Apache	20
A.1	Problem mit Zeichensatzdarstellung	20

Kapitel 1

Einführung

Jeder Download von Software oder anderen Dateien aus dem Internet birgt die Gefahr, daß sich darunter infizierte Dateien befinden können, die – geöffnet oder auf der lokalen Workstation ausgeführt – zu einer Infektion des Systems führen und sich von dort aus im lokalen Netzwerk ausbreiten.

Eine Kontamination per Email ist zur Zeit mit großem Abstand die größte Infektionsquelle. Es nützt daher wenig, sich ausschließlich auf eine Absicherung gegen eine Download-Infektion zu konzentrieren. Die wichtigste Schritt zu einem sicheren Netzwerk ist daher nach der Absicherung der Internetverbindung durch einen Firewall die Einrichtung einer Schutzmaßnahme gegen Email-Viren. Eine bewährte – wenn auch nicht die einzige – Methode ist `amavisd-new`, der auch gleich mit dem SPAM-Filter `Spamassassin` verbunden werden kann.

Trotzdem: die Infektionsgefahr beim Websurfen ist durchaus vorhanden. Dieses Dokument soll dazu beitragen, mit OpenSource-Mitteln auch den Webzugriff gegen Vireninfectionen abzusichern. Dazu werden zunächst unterschiedliche Lösungsansätze vorgestellt und ihre Vor- und Nachteile diskutiert.

In den nachfolgenden Kapiteln werden dann zwei Lösungen im Praxiseinsatz beschrieben.

1.1 Grundprinzip

Über eines muß man sich von Anfang an im Klaren sein, wenn man den Datenstrom des Websurfers nach Viren hin untersucht: es kostet nicht gerade wenig Rechenzeit, und damit Systemperformance. Ein gescannter Datenstrom bietet nicht die gleiche Zugriffsgeschwindigkeit wie ein direkter, ungefilterter Zugriff auf das Internet.

Allerdings kann man sich Gedanken darüber machen, wie man die die Performance-Einbrüche in Grenzen hält:

- ❑ *Leistungsfähige Hardware*: kann die Performance deutlich steigern.

- ❑ *Nur neue Dateien scannen:* speichert man bereits geprüfte Dateien zwischen („Caching“), müssen nur noch unbekannte Dateien geprüft werden, was bei wiederholtem Zugriff auf die gleichen Dateien einen erheblichen Performance-Gewinn bedeutet. Nachteil allerdings ist, da bei ganz neuen Viren eine infizierte Datei im Cache verbleibt, weil der Virens Scanner erst nach dem Download aktualisiert wurde.
- ❑ *Sichere Dateien nicht scannen:* Dateien oder Dateitypen, die als sicher eingestuft werden, werden vom Virens can ausgeschlossen. Das birgt prinzipiell die Gefahr in sich, daß über eine Fehleinschätzung, was eine „sichere Datei“ ist, sich doch eine infizierte Datei einschleicht. Deshalb muß man die Liste der „sicheren Dateien“ klein halten. Eine Überprüfung auf den MIME-Typ liefert generell eine wesentlich höhere Treffersicherheit als eine Überprüfung auf die nackte Dateieindung.

In der Praxis kombiniert man in der Regel alle drei Prinzipien, dabei bringt sicher der Einsatz eines Caching-Proxy vor einer Scan-Einheit den meisten Gewinn, denn bereits gespeicherte Dateien können direkt geliefert werden, der Download sowie die Entscheidung, ob es sich um eine „sichere Datei“ handelt und der Virens can wegfallen kann, muß erst gar nicht getroffen werden. In Abbildung 1.1 ist das Grundprinzip für den Virens can beim Websurfen dargestellt.

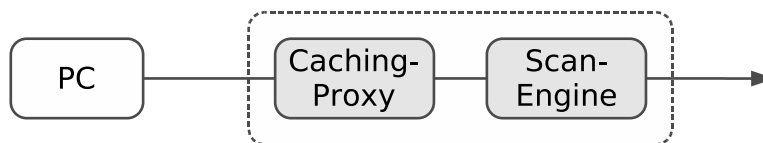


Abbildung 1.1: Grundprinzip für den Proxyscan

Der Anwender greift von seiner Workstation aus zunächst auf dem Caching-Proxy zu. Bereits dort abgelegte Dateien sind mit lokaler Netzwerkgeschwindigkeit direkt verfügbar, eine Verzögerung durch Download oder Virens can entfällt hier.

Der Proxy wird so eingestellt, daß er nicht vorhandene Dateien immer über die Scan-Engine holt und keine andere Wege einschlägt. Die Scan-Engine überprüft immer erst, ob es sich um einen Datei-Typ handelt, der als „sicher“ gilt (in der Regel die MIME-Typen `text/html`, `text/plain`, `image/gif`, `image/jpeg` und `image/png`). Alle anderen Dateien werden erst auf Viren untersucht, bevor diese an den Browser weitergereicht werden.

In den meisten Fällen bietet es sich an, Caching-Proxy und Scan-Engine auf dem gleichen Rechner unterzubringen. Insbesondere dann, wenn die Scan-Engine aus einer Kombination mit dem Webserver Apache eingesetzt wird, läßt sich die Scan-Engine so konfigurieren, daß ein Zugriff nur von `localhost` aus möglich ist.

1.2 Virensan und Browsergeduld

Findet der Scanner einen Virus, ist es sinnvoll und nützlich, den Anwender zu informieren. Nur ist das nicht so einfach. Angenommen, der Anwender lädt eine gezipptes Archiv herunter. Der Webserver meldet dem Browser den MIME-Typ `application/zip`. Bei anderen Dateien eben den passenden MIME-Typ. Findet der Scanner einen Fehler, muß er eine Info an den Anwender senden, die dieser im Browser lesen kann und nicht abspeichert, also etwa eine HTML-Seite. Eine HTML-Seite hat aber den MIME-Typ `text/html`. Der MIME-Typ wird im HTTP-Header vor der eigentlichen Datei übertragen. Die Scan-Engine muß also zuerst wissen, ob die Datei in Ordnung ist, bevor sie entscheidet, ob der Anwender die Datei bekommt, oder eine Fehlermeldung. Der Download muß also abgeschlossen und die Datei vollends gescannt sein. In der Zwischenzeit wartet der Browser ohne eine Antwort, was bei großen Dateien oft zu lange dauert. Der Browser denkt, die Verbindung ist abgebrochen, und beendet die Verbindung zum Proxy.

Die Lösung für das Timeout-Problem ist ein Kompromiß, den man akzeptieren muß, weil es keine andere Lösung gibt. Um den Timeout zu verhindern, sendet die Scan-Engine in regelmäßigen Abständen ein paar Bytes an den Anwender, damit der Browser nicht die Geduld verliert. Findet die Scan-Engine allerdings später einen Virus, gibt es keine Möglichkeit mehr, den Anwender zu benachrichtigen. Die Verbindung wird einfach abgebrochen.

Idealerweise setzt man daher eine Scan-Engine ein, die erst nach dem Ablauf einer bestimmten Zeit die ersten Datenbytes sendet. Solange das nicht geschehen ist, bleibt der Scan-Engine immer noch die Möglichkeit, den Anwender über einen vorhandenen Virus zu informieren. Setzt man dann noch einen Browser (wie Mozilla) ein, der längere Wartezeiten jenseits einer Minute in Kauf nimmt, kann man mit dem Kompromiß ganz gut leben. Bei einer leistungsfähigen Internetanbindung und nicht allzugroßen Dateien ist das recht praktikabel.

1.3 Übersicht über die unterschiedlichen Lösungen

Die nachfolgende Übersicht erhebt keinen Anspruch auf Vollständigkeit. Über die Homepages von Squid und dem OpenAntiVirus-Projekt findet sich Linkseiten auf weitere Projekte:

<http://www.squid-cache.org/related-software.html>
<http://www.openantivirus.org/projects.php>

Mit Ausnahme des eigentlichen Virensanners sollte die aufgeführte Software im wesentlichen „frei“ im Sinne des OpenSource-Begriffes sein. Eine genaue Auskunft bietet die jeweilige Homepage der beschriebenen Software.

1.3.1 squid-vscan

<http://www.openantivirus.org/>

Prinzip

squid-vscan ist ein Patch für Squid 2.3. Zum Einsatz kommt der OpenAntiVirus Scanner (OAV).

Voraussetzungen

- Squid 2.3
- Patch für Squid 2.3
- OpenAntiVirus Scanner

Vorteile

- Direktes Plugin in Squid
- Fertig übersetzter, gepatchter Squid 2.3 als RPM (SuSE 7.3) verfügbar.

Nachteile

- Veraltet. Weder OpenAntiVirus ist auf dem neuesten Stand, noch gibt es einen Patch für neuere Squid-Versionen. Die Aktivitäten scheinen eingestellt zu sein.
- Bei aktuellen Distributionen muß Squid selbst gepatcht und kompiliert werden.

1.3.2 Internet Content Adaption Protocol ICAP

www.i-cap.org/home.html

Für den Squid-Client:

<http://squid.sourceforge.net/icap/>

Prinzip

Das Internet Content Adaption Protocol ICAP ist ein relativ neues Protokoll, daß von verschiedenen Herstellern von Virenscannern aufgegriffen wurde. Squid läßt sich dabei als Caching-Proxy und ICAP-Client einsetzen, als ICAP-Server dient ein kommerzieller Virenschanner, der dieses Protokoll unterstützt.

Voraussetzungen

- Squid 2.5 oder neuer
- Patch ICAP-Client für Squid (ggf. ab 3.0 nicht mehr erforderlich)
- ICAP-Server mit Virenschanner-Funktionalität, oder Virenschanner, der über eine ICAP-Server-Funktionalität verfügt.

Vorteile

- ICAP wird zunehmend von Herstellern von Antivirus-Software unterstützt, dadurch fast freie Wahl des Antivirus-Scanners.
- Der ICAP-Client für Squid wird offiziell vom Squid-Team gepflegt und wird in eine der nächsten Releases integriert.

Nachteile

- Derzeit (Squid 2.5) noch ein Patch von Squid erforderlich.
- kommerzieller Virenschanner erforderlich.
- Keine Erfahrungen mit Performance und Konfiguration.

1.3.3 Viralator

<http://viralator.sourceforge.net/>

Prinzip

Squid wird mit einem Redirector konfiguriert. Fordert der User eine URL an, wird diese an den Redirector übergeben. Das Redirector-Skript lädt die Datei in ein eigenes Verzeichnis, die Datei wird dort nach Viren gescannt. Ist die Datei infiziert, erhält der User eine Warnmeldung, ist die Datei „sauber“, wird ein Redirect gesetzt, der Browser lädt die Datei dann von der neuen Lokation (Apache, der auf das Verzeichnis Zugriff hat, in dem der Scan stattfand).

Voraussetzungen

- Squid
- Als Redirector: Squirm oder SquidGuard
- wget
- Apache
- Virenschanner

Vorteile

- Unterstützt sehr viele Virenschanner
- Läuft mit Board-Methoden von Squid

Nachteile

- Aufwendig zu konfigurieren
- Unterscheidung der Dateitypen nur nach der Endung

1.3.4 Apache::ProxyScan

<http://search.cpan.org/dist/ProxyScan/>

Prinzip

Apache wird im Proxy-Modus als Parent Cache zu Squid eingesetzt. Apache::ProxyScan ist ein Perlmodul, daß sich in den Proxy-Modus bei Apache einklinkt.

Voraussetzungen

- Apache 1.3 im Proxy-Modus
- mod_perl für Apache
- Apache::ProxyScan
- Virens Scanner

Vorteile

- Unterstützt verschiedene Virens Scanner
- Relativ einfache Konfiguration
- Erkennung von „sicheren“ Dateien nach Datei-Endung und/oder MIME-Typ.

Nachteile

- langsamer als andere Lösungen, da externe Perlskripte aufgerufen werden.
- Die MIME-Typ-Erkennung funktioniert nicht immer korrekt, es werden dann auch „sichere“ Dateien gescannt, was den Vorgang zusätzlich verlangsamt.

1.3.5 Apache2 und mod_clamav

http://software.othello.ch/mod_clamav/
<http://www.clamav.net>

Prinzip

Apache2 wird im Proxy-Modus als Parent für Squid betrieben. Das Modul `mod_clamav` wird über den bei Apache2 neuen Filtermechanismus eingebunden. `mod_clamav` benutzt entweder direkt die Library des Virenschanners Clamav (dann muß kein externes Programm aufgerufen werden), oder kontaktiert den Clamav-Daemon `clamd`.

Voraussetzungen

- Apache2 mit allen `mod_proxy*`-Modulen (Standard)
- `mod_clamav`
- Clamav-Library `libclamav1`

Vorteile

- Direkte Einbindung in Apache, Aufruf externer Skripte oder Programme entfällt.
- Funktionaler OpenSource-Virenschanner Clamav
- Erkennung von „sicheren Dateien“ über MIME-Typen, URLs und Pattern möglich.
- Ausführliche Dokumentation
- aktuell

Nachteile

- Einschränkung auf Clamav als Virenschanner
- `mod_clamav` nicht im Standard-Lieferumfang, muß selbst übersetzt werden.

1.3.6 Apache2 und `mod_vscan`

Prinzip

Gleiches Prinzip wie bei `mod_clamav`, es wird eben nur statt Clamav der ebenfalls freie Scanner OpenAntiVirus (OAV) benutzt.

Voraussetzungen

- Apache2 mit allen `mod_proxy*`-Modulen (Standard)
- `mod_vscan`
- OpenAntiVirus Scanner und Library

Vorteile

- Direkte Einbindung in Apache, Aufruf externer Skripte oder Programme entfällt.
- Erkennung von „sicheren Dateien“ über MIME-Typen.

Nachteile

- Einschränkung auf OpenAntiVir als Virenschanner. Dieser scheint derzeit nicht besonders aktiv weiterentwickelt zu werden und hinkt in bezug auf aktuelle Pattern anderen Scannern stark hinterher.
- mod_vscan nicht im Standard-Lieferumfang, muß selbst übersetzt werden.
- Keine aktive Entwicklung des Moduls?

Kapitel 2

Apache::ProxyScan

Warnhinweis: die nachstehende Beschreibung geht auf die notwendige Konfiguration ein, die erforderlich ist, um den Virensan einsetzen zu können. Für eine sichere Konfiguration von Squid und Apache ist der Administrator selbst verantwortlich!

2.1 Vorbereitungen

Zunächst wird Apache 1.3 einschließlich des Perl-Modules `mod_perl` benötigt. Beide sind bei allen gängigen Linux-Distributionen vorhanden, `mod_perl` muß aber meist explizit für die Installation ausgewählt werden.

Das Perl-Modul `Apache::ProxyScan` ist in keiner Distribution vorhanden und muß daher über das CPAN (Comprehensive Perl Archiv Network) geholt und installiert werden. Dazu gibt es zwei Möglichkeiten: die manuelle Installation oder die CPAN-Variante.

Manuelle Installation

Die Module finden sich unter <http://www.cpan.org>.

Als User `root` Archiv auspacken, übersetzen und installieren:

```
% cd /usr/local/src/Perl
% tar xvzf Apache-ProxyScan-0.31.tar.gz
% cd Apache-ProxyScan-0.31/
% Perl Makefile.PL
% make ; make check ; make install
```

Gegebenenfalls fehlende Perl-Module sollten von der Distribution nachinstalliert werden, soweit vorhanden, andernfalls kann man diese auf den hier beschriebenen Weg natürlich ebenfalls nachinstallieren.

CPAN-Automatik

```
perl -MCPAN -e "install Apache::ProxyScan"
```

Perl holt sich hier automatisch fehlende Module aus dem CPAN-Archiv.

Konfigurationsschema

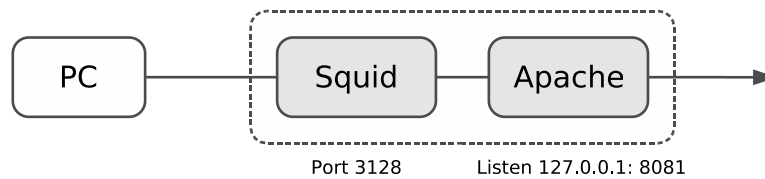


Abbildung 2.1: Squid-Apache-Kombination

Das verwendete Konfigurationsschema ist in Abbildung 2.1 dargestellt. Squid und Apache sind hier auf demselben Host installiert. Squid ist für den Standard-Port 3128 konfiguriert, den der Browser benutzt. Apache lauscht ausschließlich auf `127.0.0.1:8081`, kann also nur vom lokalen Host aus erreichbar werden.

2.2 Squid-Konfiguration

Squid wird angewiesen, ausschließlich einen Parent Cache zu befragen. Unterläßt man dies, bemüht sich Squid selbst um die angefragten Seiten. Der Virenschutz würde dadurch umgangen werden.

```
cache_peer localhost parent 8081 0 \
           default no-query no-digest
read_timeout 90 minutes
never_direct allow all
```

Der Timeout ist erforderlich für große Dateien, um zu verhindern, daß Squid nicht zwischendurch die Geduld verliert und den Download abbricht.

2.3 Apache-Konfiguration

Die nachstehende Datei `/etc/apache/httpd.conf` ist keine vollständige Konfigurationsdatei, sondern beschreibt nur die für einen Virenskan notwendigen Parameter.

```

# Auszug aus /etc/apache/httpd.conf
...
Listen 127.0.0.1:8081

# -- Modules
# z.B. <path> /usr/lib/apache/1.3/
...
LoadModule mime_magic_module <path>/mod_mime_magic.so
LoadModule mime_module <path>/mod_mime.so
...
LoadModule proxy_module <path>/libproxy.so
...
LoadModule perl_module <path>/mod_perl.so
...

# -- Proxy for ProxyScan

ProxyRequests On

PerlTransHandler +Apache::ProxyScan
PerlSetVar ProxyScanScanner "/usr/local/bin/clamav.pl"
PerlSetVar ProxyScanTempDir /var/cache/virus/dl/
PerlSetVar ProxyScanPredeliverSize 102400
PerlSetVar ProxyScanTrustedMIME "image/* text/html"
PerlSetEnv SCAN_TMP /var/cache/virus/av/

<Directory proxy:*>
  Options Indexes FollowSymLinks MultiViews
  IndexOptions +FancyIndexing +NameWidth=*

  <Limit CONNECT GET POST>
    Order allow,deny
    Allow from 127.0.0.0/255.0.0.0
  </Limit>
  <LimitExcept CONNECT GET POST>
    Order deny,allow
    Deny from all
  </LimitExcept>
</Directory>

```

Hinweise

- Perl-Modul*: Wichtig ist, das Perl-Modul erst nach dem Proxy-Modul zu laden.
- ProxyRequests On*: latente Sicherheitslücke jeder, der den Proxymodus verwenden kann, kann hierüber auch Emails verschicken.

- ❑ *PerlTransHandler*: das + sorgt dafür, daß das Modul bereits bei Start von Apache geladen wird.
- ❑ *ProxyScanScanner*: Wrapper-Skript für Virens Scanner, hier Clamav (wird in der Source-Distribution von Apache::ProxyScan mitgeliefert).
- ❑ *ProxyScanTempDir*, *SCAN_TMP*: beide Verzeichnisse müssen existieren und für den User clamav les- und schreibbar sein.
- ❑ *ProxyScanPredeliverSize*: liefert schon mal vorab einige Bytes aus, damit der Browser nicht die Geduld verliert. Allerdings ist dann keine Benachrichtigung des Users im Virensfall möglich.
- ❑ *ProxyScanTrustedMIME*: Angabe der „sicheren“ MIME-Typen. Funktioniert möglicherweise nicht perfekt, alternativ kann man die weniger sichere Variante *ProxyScanTrustedExtension* verwenden.
- ❑ *Directory proxy:**: Umgebung für Definitionen ausschließlich bei Proxy-Zugriff.
- ❑ *Limit*: die beiden Direktiven *Limit* und *LimitExcept* beschränken den Zugriff auf localhost und auf die Methoden GET, POST, und CONNECT.

Weitere Infos mit man `Apache::ProxyScan` (nach der Installation des Moduls).

Kapitel 3

Apache2 Modul: mod_clamav

Warnhinweis: die nachstehende Beschreibung geht auf die notwendige Konfiguration ein, die erforderlich ist, um den Virenskan einzusetzen zu können. Für eine sichere Konfiguration von Squid und Apache ist der Administrator selbst verantwortlich!

3.1 Vorbereitungen

STICHWORT: APACHE 2.0, MOD_CLAMAV, LIBCLAMAV1

Neben einer Grundinstallation von Apache 2.0 ist die Installation der Clamav-Bibliothek `libclamav1` erforderlich. Diese wird entweder als eigenes Paket angeboten, oder ist im Clamav-Paket der Distribution enthalten. Binary Packages gibt es auch auf der Homepage von Clamav:

<http://www.clamav.net>

Das Modul `mod_clamav` kann neben einem Builtin-Scan über die Clamav-Bibliothek auch den Clamav-Daemon `clamd` als externen Scanner aufrufen. In diesem Falle muß `clamd` je nach Distribution zusätzlich installiert werden. Für die Aktualisierung der Virensignaturen ist bei Clamav `freshclam` zuständig. `freshclam` wird entweder über `/etc/crontab` regelmäßig aufgerufen oder als Daemon installiert.

mod_clamav installieren

Das Modul erhält man unter folgender URL:

http://software.othello.ch/mod_clamav/

Nach dem Entpacken ruft man `configure`, daß als einzigen Parameter ein Werkzeug von Apache2 benötigt (darüber wird die Konfiguration des installierten Apache2-Servers bestimmt):

```
% /cd /usr/local/src
% tar xvzf mod_clamav-0.20.tar.gz
```

```
% cd mod_clamav-0.20
% ./configure --with-apxs=/path/to/apache2/bin/apxs2
% make ; make install
```

Möglicherweise schlägt `make` fehl, weil einige Include-Dateien fehlen. Dann ermittelt man, um welche Dateien es sich handelt. Auf Debian/Sarge zum Beispiel erwartet `make` die Include-Files alle in `/usr/include`, allerdings befinden sich alle `apr`-Include-Files in einem Unterverzeichnis `apr-0`. Das läßt sich im Template `Makefile.in` korrigieren:

```
mod_clamav_la_CPPFLAGS = -I`${APXS}` -q INCLUDEDIR ` \
                        -I/usr/include/apr-0
```

Mit `-I` lassen sich zusätzliche Include-Verzeichnisse angeben. Ein erneuter Aufruf `./configure` führt zu einem korrekten Makefile und das Übersetzen gelingt jetzt.

Schlägt ein `make install` fehl, ist das nicht weiter schlimm. Es wird nur das Modul im Modul-Verzeichnis von Apache2 benötigt:

```
cp mod_clamav.so /usr/lib/apache2/modules/.
```

Das Verzeichnis für Debian/Sarge ist `/usr/lib/apache2/modules/`, bei anderen Distributionen kann das ein anderes Verzeichnis sein.

Konfigurationsschema

Das Konfigurationsschema ist identisch mit dem bei Apache::ProxyScan (siehe Abbildung 2.1).

3.2 Squid-Konfiguration

Squid wird wieder angewiesen, Apache als Parent Cache zu befragen:

```
cache_peer localhost parent 8081 0 \
           default no-query no-digest
read_timeout 90 minutes
never_direct allow all
```

3.3 Apache-Konfiguration

Für die Apache2-Konfiguration wird davon ausgegangen, daß alle Konfigurationsdateien im Verzeichnis `/etc/apache2/conf.d` verwendet werden. Im zentralen Konfigurationsfile `/etc/apache2/apache2.conf` muß daher folgende Anweisung stehen:

```
Include /etc/apache2/conf.d
```

Der Vorteil bei dieser Variante ist, daß man eine eigene Konfigurationsdatei für `mod_clamav` anlegen und pflegen kann, die vom Softwarepaket nicht angefaßt wird und somit Update-fest ist.

Um zu verhindern, daß der Apache über daß Netzwerk erreichbar ist, trägt man in `/etc/apache2/ports.conf` (Debian/Sarge) folgende Direktive ein:

```
Listen 127.0.0.1:8081
```

`/etc/apache2/conf.d/clamav.conf`

```
# -- load modules
# -- <mpath> = /usr/lib/apache2/modules (Debian/Sarge)

LoadModule proxy_module <mpath>/mod_proxy.so
LoadModule proxy_ftp_module <mpath>/mod_proxy_ftp.so
LoadModule proxy_http_module <mpath>/mod_proxy_http.so
LoadModule proxy_connect_module \
    <mpath>/mod_proxy_connect.so
LoadModule clamav_module <mpath>/mod_clamav.so

# -- mod_proxy

<IfModule mod_proxy.c>

    ProxyRequests On

    <Proxy *>
        # -- using Apache2 filter mechanism
        SetOutputFilter CLAMAV

        # -- restricted access
        Order deny,allow
        Deny from all
        Allow from 127.0.0.1
    </Proxy>
```

```

ProxyVia On

</IfModule>

# -- mod_clamav

<IfModule mod_clamav.c>

    # ClamavExtendedLogging On
    # LogFormat "%t %!304{clamav:status}n \
    #   %{Content-type}o \
    #   %{clamav:virusname}n request=\"%r\", \
    #   status=%>s, sent=%!304b, delay=%!304D" \
    #   clamav_stats
    # CustomLog /var/log/apache2/scan.log clamav_stats

    # -- Clamav directives

    ClamavTmpdir /var/cache/virus/dl
    ClamavDbdir /var/lib/clamav
    ClamavReloadInterval 1800
    ClamavSafetypes image/gif image/jpeg image/png
    ClamavSafeTypes image/x-ico
    ClamavSafetypes text/html text/plain

    <Location /clamav>
        # -- builtin status report
        SetHandler clamav

        Order deny,allow
        Deny from all
        Allow from 127.0.0.1

    </Location>

</IfModule>

```

Hinweise

- ❑ *ProxyRequests On*: schaltet die Proxy-Funktionalität für Apache2 ein. Das ist eine potentielle Sicherheitslücke, da der Server jetzt als Relay dienen kann. Schalten Sie die Direktive erst an, wenn Sie Apache2 und Ihren Server insgesamt abgesichert haben.
- ❑ *<Proxy *>*: Umgebung für die Proxy-Funktionalität.

- ❑ *SetOutputFilter CLAMAV*: weist Apache2 an, `mod_clamav` als Output-Filter (neu seit Apache 2.0) zu verwenden.
- ❑ *ClamavExtendedLogging On*: füllt zusätzliche Tabellen für das Logging. Die Tabelle `clamav:status` liefert die Werte `passed`, `bypassed`, `aborted`, `INFECTED` or `failed`. Mehr dazu in der Online-Hilfe unter http://software.othello.ch/mod_clamav/
- ❑ *LogFormat*: Ohne eine `LogFormat`-Angabe würde die Information, die mit `ExtendedLogging` erzeugt wird, nicht sichtbar. Der String muß auf einer einzigen Zeile stehen.
- ❑ *ClamavTmpdir*: temporäres Verzeichnis zum Scannen von Dateien.
- ❑ *ClamavDbdir*: Datenbankverzeichnis von Clamav mit den Signaturen.
- ❑ *ClamavReloadInterval*: Interval in Sekunden, nach denen das Modul `mod_clamav` die Signatur-Datenbank neu liest.
- ❑ *ClamavSafetypes*: Angabe von „sicheren“ MIME-Typen
- ❑ *SetHandler clamav*: Eingebaute Statistik. Um an diese Statistik zu gelangen, muß der Browser auf den Proxy (Squid, Port 3128) zugreifen und darf `localhost` nicht in der Ausschlußliste stehen haben (dann wird `localhost` ebenfalls an den Proxy weitergereicht und greift dann lokal am Proxy zu). Die URL ist dann:
<http://localhost:8081/clamav/>

Anhang A

Apache

A.1 Problem mit Zeichensatzdarstellung

Breibt man Apache im Proxy-Mode, werden externe Seiten geladen, über die Apache keine Kontrolle hat. Ein Default-Zeichensatz ist daher nicht sinnvoll.

Sowohl Apache 1.3 als auch Apache 2.0 kennen die Direktive

```
AddDefaultCharset On|Off|<charset>
```

Die Direktive weist Apache an, den in der Datei enthaltenen Zeichensatz (META-Tag) zu ignorieren und mit dem gesetzten Zeichensatz explizit zu überschreiben. Surft man zum Beispiel bei `www.google.de`, erhält man seltsame Zeichen für die Umlaute. Das von Google gesetzte META-Tag enthält in der Regel UTF-8, was Apache mit gesetzter Direktive einfach ignoriert.

In einigen Distributionen ist im Konfigurationsfile entweder die Direktive explizit gesetzt (z.B. Debian/Sarge Apache 2.0), oder einkompiliert (Debian/Woody Apache 1.3).

Tritt das oben beschriebene Problem auf, kann man den Default-Zeichensatz einfach abschalten:

```
AddDefaultCharset Off
```