

Ein Whitepaper zur effektiven Absicherung eines LAMP-Server

Patrick Schmid (encodingit.ch)

17. Mai 2011

1 Vorwort

Dieses Whitepaper beschäftigt sich mit der Absicherung eines auf Linux basierenden Servers. Im den Tests wurde ein Debian-basiertes System verwendet, wodurch die Namen von einzelnen Paketen oder Pfade zu Konfigurationsdateien minimal abweichen können.

Auch gilt es zu erwähnen, dass es sowas wie die totale Sicherheit nicht gibt, denn in der IT Security kann man nur von Wahrscheinlichkeit sprechen! Jedoch aber, aus der Kombination von mehreren Sicherheitsvorkehrungen kann die Wahrscheinlichkeit eines Angriffes so verringert werden, dass sich stark am Unmöglichen annähert.

Der Autor dieses Dokumentes ist Patrick Schmid, ein nach LPIC zertifizierter Linux System Engineer und Verfechter von Opensource.

Mehr Informationen gibt es unter <http://encodingit.ch>.

Inhaltsverzeichnis

1	Vorwort	2
2	Identifizieren	5
2.1	netstat	5
2.2	nmap	5
3	Vorbereitung	6
3.1	Aufräumen	6
3.2	Absichern	6
3.3	Aktualisieren	6
3.4	Passwort	7
4	SSH	8
4.1	Root Login verbieten	8
4.2	Schlüssel	8
4.3	Non-Standard	9
4.4	Einschränken	9
5	MySQL	11
5.1	Remote Access	11
5.2	Non-Standard	11
5.2.1	Passwort	11
5.2.2	Port	12
5.3	Aufräumen	12
5.3.1	Benutzer	12
5.3.2	Datenbanken	12
6	Apache	13
6.1	SSL	13
6.2	Zugriff	14
7	PHP	15
7.1	Einschränken	15
8	Abblocken	16
8.1	iptables	16

Inhaltsverzeichnis

9	Hilfsmittel	17
9.1	Anklopfen	17
9.2	Verbannen	18
9.3	Mod Security	19

2 Identifizieren

Bevor man überhaupt mit dem Absichern eines Servers beginnen kann, muss man zuerst wissen, gegen was man überhaupt sichern soll. Somit ist es essentiell zu wissen, was auf dem Server tatsächlich alles aktiv ist. Das Augenmerk liegt dabei auf den offenen Ports und aktuellen Verbindungen!

2.1 netstat

Unter Verwendung des Tools netstat kann man sich eine Liste mit allen aktiven Verbindungen, samt Ports ausgeben lassen:

```
netstat -pantlu | grep -v 127.0.0.1 | grep -v 0.0.0.0
```

Hierzu muss man natürlich seinen eigenen Server kennen. Läuft so zum Beispiel ein Webserver, so ist ein aktiver Port 80 oder 443 normal. Da es sich bei dem Betriebssystem um ein Linux handelt, ist auch ein offener Port 22 für SSH nicht verdächtig. Erst wenn unbekannte Ports in der Liste erscheinen, sollte man aufmerksam werden, denn dies könnte bereits ein potenzieller Angreifer sein!

Nun aber liefert netstat nur eine aktuelle Übersicht über den aktuellsten Stand. Sprich, wenn vor einiger Zeit mal FTP installiert, aber seither nicht mehr genutzt, so gibt es keine aktive Verbindung dazu. Jedoch ein Angreifer könnte über den in Vergessenheit geratenen FTP-Server über Port 21 in das System eindringen und dieses kompromittieren!

2.2 nmap

Also muss das System noch "von aussen" mittels nmap geprüft werden:

```
nmap -PN localhost
```

Dadurch erhält man eine Liste mit den Ports und den erkannten Diensten, welche einem Angreifer als möglicher Einfallspunkt dienen könnten. Diese gilt es nun, soweit als möglich, abzusichern!

3 Vorbereitung

3.1 Aufräumen

Wie nmap uns gezeigt hat, läuft auf unserem Server noch eine Instanz von FTP auf dem Port 21. Jedoch wird dies gar nicht mehr benötigt, da die Webseite nur via SSH gewartet wird!

Dies ist eines der grössten Probleme und eine der besten Angriffspunkte, denn in Vergessenheit geratene Dienste und Pakete werden auch nicht mehr gepflegt und liegen dadurch in veralteten und anfälligen Version unbemerkt auf dem Server!

Das Beste ist es also, diese Pakete gleich vom Server zu löschen. Oder, falls das nicht möglich ist, weil man noch auf die Daten angewiesen ist, wenigstens den Dienst am automatischen starten hindern. Dazu wird der entsprechende Service unter `/etc/init.d/` aufgespürt und zu Beginn ein `«exit 0»` eingefügt. Oder, falls der Service nicht als eigener Dienst startet, so wird die Datei `/etc/inetd.conf` geöffnet und der entsprechende Eintrag auskommentiert oder gelöscht! Nach einem Neustart ist der Dienst nun nicht mehr verfügbar!

3.2 Absichern

Das `/tmp`-Verzeichnis, eine Art künstlicher, flüchtiger Speicher ist sehr beliebt bei potenziellen Angreifern, da, nachdem der Server abgestürzt ist oder neugestartet wurde, alle Spuren automatisch gelöscht werden. Deshalb wird dieses Verzeichnis speziell geschützt. Dazu wird die Datei `/etc/fstab` mit folgendem Eintrag ergänzt:

```
none /tmp tmpfs nodev , nosuid , noexec 0 0
```

3.3 Aktualisieren

Die IT ist ein sich stetig weiterentwickelnder Organismus. Dieser entwickelt sich viel schneller als alles andere, weshalb es für uns auch zwingend nötig ist, stets auf dem neusten Stand zu sein. Denn nur, wenn man das eigene Betriebssystem und deren Software und Dienste stets auf dem neusten Stand hält, macht es überhaupt Sinn, irgendetwas abzusichern! Um stets die neusten Version zu installieren, kann man die bereits vorinstallierten Paketmanager wie `apt-get`, `yum` oder `zypper` verwenden, um schnell und unkompliziert zu aktualisieren.

3.4 Passwort

Das Passwort ist die wichtigste Komponente für einen sicheren Server! Denn ist das Passwort schwach, so kann es leicht erraten werden... Auch sollte das Passwort nicht aus dem Wörterbuch stammen, damit es nicht durch allfällige Brute-force-Angriffe geknackt wird. Deshalb empfiehlt sich ein Passwort aus mehr als 8 Zeichen, mit Gross- und Kleinbuchstaben und Zahlen. Im Idealfall auch noch Sonderzeichen. Um es sich aber trotzdem noch merken zu können, kann man auf ein paar Tricks zurückgreifen.

So gibt es Zahlen, die ersetzten Buchstaben, da diese sehr ähnlich aussehen. Beispiele dazu wären 1 für l, 4 für A, 5 für S, 0 für O und mehr. Oder man fasst aus einem Satz jeweils die ersten oder letzten Buchstaben zu einem Wort zusammen. Und im Idealfall - man mischt beides: So ergibt sich ein Passwort «1,d4sP,bu!».

Abkürzung	Zuordnung
1	Ich
,	,
d	das
4	absolut
s	sichere
P	Passwort
,	,
b	bin
u	unknackbar
!	!

Schwer zu merken, doch wenn man weiss, dass dahinter der Satz «Ich, das absolut sicherer Passwort, bin unkackbar!» steht, so kann man sich das Problemlos merken.

4 SSH

Nun, da die Grundlagen geschaffen sind, kann der Server abgesichert werden. Als erstes kommt dabei SSH, da ein Grossteil der Angriffe über dieses Protokoll ablaufen.

4.1 Root Login verbieten

Da ein root-Benutzer, der sich von überall her auf dem Server einloggen kann, sehr viel Macht besitzen würde, sollte dies unterbunden werden. Dies hat auch den Vorteil, dass der Angreifer neben dem richtigen Passwort zuerst auch noch den Benutzernamen herausfinden muss, da der Standard root ja gesperrt ist. Dazu muss die Konfigurationsdatei von SSH unter `/etc/ssh/sshd_config` angepasst werden. Der Eintrag «PermitRootLogin» wird dabei mit einem «No» ergänzt:

Port	22
Protocol	2
PermitRootLogin	no
RSAAuthentication	no
PubkeyAuthentication	no
PasswordAuthentication	yes
ChallengeResponseAuthentication	no
X11Forwarding	yes
ClientAliveInterval	30

4.2 Schlüssel

Nun aber kann es immer mal sein, dass ein Passwort geknackt werden kann. Die Gründe dafür sind hier nicht von Bedeutung. Sollte auf unserem Server das Passwort geknackt werden können, so kann sich ein Angreifer problemlos via SSH authentifizieren und unser Server wäre wiederum kompromittiert! Also greifen wir auf Schlüssel oder Keys zurück. Dies ist eine spezielle Datei, welche, zusätzlich zum Passwort für ein erfolgreiches Login benötigt wird. Somit muss der Angreifer nicht nur das Passwort knacken, sondern sich auch mein Key aneignen, was die Sache doch erheblich erschwert...

Also erzeugen wir zuerst unsere eigenen Key:

```
ssh-keygen -t dsa
```

Nun werden wir nach einem Passwort gefragt, und unser Key wird erstellt im Ordner `/.ssh`. Von da aus, kopieren wir ihn nun auf den Server:

```
scp ~/.ssh/id_dsa .pup SERVER:/authorized_keys2
```

4 SSH

Nun muss wiederum die Konfigurationsdatei von SSH angepasst werden:

Port	22
Protocol	2
PermitRootLogin	no
RSAAuthentication	no
PubkeyAuthentication	yes
PasswordAuthentication	no
ChallengeResponseAuthentication	no
X11Forwarding	yes
ClientAliveInterval	30

Von nun an ist es nur noch möglich, sich mittels Key und dem dazu passenden Passwort einzuloggen. **Aber Vorsicht: Ohne Key läuft nichts, deshalb sollte dieser gut verwahrt werden!**

4.3 Non-Standard

Ausserdem kann es sehr nützlich sein, wenn man mit seinem Server nicht dem Standard entspricht. So gibt es im Netz viele automatisierte Angriffe, welche den Standardport von SSH 22 mit den hundert gängigsten Passwörtern prüfen und dann weiterziehen. Wenn ich nun aber SSH von Port 22 auf Port 33 lege, so kann ich den meisten dieser Angriffe ein Strich durch die Rechnung machen! Also sieht unsere Konfiguration neu so aus:

Port	33
Protocol	2
PermitRootLogin	no
RSAAuthentication	no
PubkeyAuthentication	yes
PasswordAuthentication	no
ChallengeResponseAuthentication	no
X11Forwarding	yes
ClientAliveInterval	30

Will man sich nun einloggen, so muss man neben dem Namen auch noch über die Option `-p` den Port 33 mitgeben:

```
ssh SERVER -p 33
```

4.4 Einschränken

Standarmässig kann sich jeder auf dem System eingerichtete Benutzer via SSH remote einloggen. Dies ist natürlich wiederum ein Sicherheitsleck, denn umso grösser die Auswahl, umso grösser die Chance einen Benutzernamen zu erraten. Also empfiehlt es sich, sich auf einen Benutzer für das Remotemanagement zu beschränken. Doch dieser sollte auf keinen Fall einen offensichtlichen Namen wie `«ssh»` oder ähnliches erhalten! Hat man sich also zum Beispiel für den Benutzernamen `«patrick»` entschieden, so wird das Login auf diesen beschränkt:

4 SSH

Port	33
Protocol	2
PermitRootLogin	no
RSAAuthentication	no
PubkeyAuthentication	yes
PasswordAuthentication	no
ChallengeResponseAuthentication	no
X11Forwarding	yes
ClientAliveInterval	30
AllowUsers	patrick

5 MySQL

MySQL gehört zu den Standard-Services auf einem LAMP-Server. Doch kann auch hier in der Konfiguration so einiges verschärft werden.

5.1 Remote Access

Wenn man nicht gerade eine Serverfarm zur Verfügung hat, so läuft MySQL, PHP und Apache meist auf dem selben Server. Somit muss nur ein lokaler Zugriff (von PHP zu MySQL) erfolgen, um die Datenbank abzufragen. Da es also gar nicht von Nöten ist, dass je jemand von ausserhalb des lokalen Servers auf die Datenbank zugreift, kann hier der «Remote Access» problemlos deaktiviert werden, wodurch man das Sicherheitslevel wieder ein bisschen erhöht. Dazu die Datei `/etc/mysql/my.cnf` wie folgt gealtert:

```
[...]
[mysqld]
user          = mysql
pid-file      = /var/run/mysqld/mysqld.pid
socket        = /var/run/mysqld/mysqld.sock
port          = 3306
basedir       = /usr
datadir       = /var/lib/mysql
tmpdir        = /tmp
language      = /usr/share/mysql/english
skip-external-locking
bind-address  = 127.0.0.1
[...]
```

5.2 Non-Standard

5.2.1 Passwort

Auch wenn mittlerweile auch die Standardkonfiguration angepasst wurde, so war es für MySQL früher normal, dass der root-Benutzer standardmässig kein Passwort gesetzt hatte. Heute wird automatisch das Passwort vom lokalen Benutzer root übernommen. Jedoch ist auch dies nicht die ideale Lösung, da wenn ein Angreifer so ein Passwort knackt, so hat er sofort Zugriff auf alle Services. Deshalb ändern wir dies ab. Und auch hier gelten die normalen Anforderungen an ein Passwort:

```
mysqladmin -u root -p'ALTES.PASSWORT' password 'NEUES.PASSWORT'
```

5.2.2 Port

Nun geht es wieder darum, den automatisierten Attacken auszuweichen. Obwohl von extern eigentlich gar kein Zugriff mehr auf MySQL möglich ist, kann es immer sein, dass die Technik mal versagt, weshalb wir hier alles schützen, so gut wir es können. Es wird also wieder die Konfigurationsdatei angepasst:

```
[...]
[mysqld]
user                = mysql
pid-file            = /var/run/mysqld/mysqld.pid
socket              = /var/run/mysqld/mysqld.sock
port                = 4408
basedir             = /usr
datadir             = /var/lib/mysql
tmpdir              = /tmp
language            = /usr/share/mysql/english
skip-external-locking
bind-address        = 127.0.0.1
[...]
```

5.3 Aufräumen

5.3.1 Benutzer

Anonyme Benutzer und solche ohne gesetztem Passwort sollten auf einem produktiven System nie vorkommen. Also werden auch diese restlos entfernt:

```
DELETE FROM mysql.user WHERE User = '';
FLUSH PRIVILEGES;
```

Nun sollte man sich die verbleibenden Benutzer ansehen, und die nicht mehr gebrauchten entfernen:

```
SELECT User, Host, Password FROM mysql.user;
DELETE FROM mysql.user WHERE User='XYZ';
FLUSH PRIVILEGES;
```

5.3.2 Datenbanken

Auch hier gilt es wieder, den bekannten Benutzer root möglichst nicht zu benutzen. Also legen wir für unsere Datenbank einen eigenen Benutzer an, mit welchem wir arbeiten können:

```
GRANT ALL ON DATENBANK.* TO 'BENUTZER'@'localhost'
IDENTIFIED BY 'PASSWORT';
```

Und nicht vergessen *DATENBANK*, *BENUTZER* und *PASSWORT* durch die eigenen Daten zu ersetzen! Und gleichzeitig verliert root natürlich seine Rechte:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'root'@'localhost';
```

6 Apache

6.1 SSL

Ein System ist jeweils nur so sicher, wie seine schwächste Komponente! So kann ein Webserver auf einem Server noch so sicher konfiguriert sein, wenn man sich aus einem unsicheren Netz heraus, zum Beispiel ein öffentliches WLAN, auf dem Server einloggt, so können die Daten samt Passwörter mitgeschnitten werden. Damit das aber nicht passiert, wird auf unserem Server der Verkehr via SSL verschlüsselt. Dazu wird zuerst ein eigenes Zertifikat angelegt:

```
mkdir -p /etc/apache2/ssl
openssl req -new -x509 -days 365 -nodes -out /etc/apache2/ssl/apache.pem -
    keyout /etc/apache2/ssl/apache.pem
ln -sf /etc/apache2/ssl/apache.pem /etc/apache2/ssl/"/usr/bin/openssl x509
    -noout -hash < /etc/apache2/ssl/apache.pem'.0
chmod 600 /etc/apache2/ssl/apache.pem
```

Nun kann der Webserver auf Port 443 umgestellt werden. Es empfiehlt sich, diesen Standard beizubehalten, da sonst viele Besucher nicht mehr auf die Webseite kommen. Dazu wird in der Datei `/etc/apache2/ports.conf` der neue Port eingetragen:

```
NameVirtualHost *:80
Listen 80

<IfModule mod_ssl.c>
    Listen 443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 443
</IfModule>
```

Jetzt können die passenden Module aktiviert werden:

```
a2enmod ssl
```

Damit alles läuft, muss nur noch der passende Virtualhost erstellt werden. Dazu wird auf die Standarddatei `default-ssl` zurückgegriffen. Diese wird jedoch nicht verändert, sondern zuerst nach `<<ssl>>` kopiert.

Nur die folgenden Änderungen müssen noch durchgeführt werden:

```
NameVirtualHost *:443
<virtualhost *:443>
...
    SSLEngine On
    SSLCertificateFile /etc/apache2/ssl/apache.pem
    #SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
...
</virtualhost>
```

Nun muss dieser nur noch aktiviert, und die alten deaktiviert werden:

```
a2ensite ssl
a2dissite default
a2dissite default-ssl
/etc/init.d/apache2 restart
```

Und unsere Seite läuft verschlüsselt über ein Zertifikat. Wenn man nicht jedes mal eine Fehlermeldung erhalten möchte, so kann man entweder auf ein CAcert zurückgreifen oder sich für wenig Geld eines kaufen.

6.2 Zugriff

Je nachdem muss die Webseite auf dem eigenen Server nicht immer für alle Personen zugänglich sein. So zum Beispiel, wenn es sich um eine Intranet-Seite handelt, welche nur innerhalb des lokalen Subnets verfügbar sein soll. Auch dafür hat apache gesorgt. Wir passen den eben erstellten Virtualhost «ssl» an:

```
[...]
<Directory /var/www/>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride None
    Order deny, allow
    deny from all
    allow from 192.168.1.1/24
</Directory>
[...]
```

7 PHP

PHP bietet viele lustige Funktionen. Jedoch auch sehr viele Einfallsmöglichkeiten für einen potentiellen Angreifer, weshalb essentiell ist für einen Programmierer, so sauber und sicher wie möglich zu arbeiten und alle Benutzereingaben zu prüfen und zu parsen.

7.1 Einschränken

In PHP gibt es viele erlaubte Funktionen, welche als potentiell gefährlich eingestuft werden können. Jedoch gibt es genauso die Funktion in der Datei `/etc/php5/apache2/php.ini` diese zu sperren:

```
[...]
disable_functions = "system , passthru , exec , popen , proc_close , proc_get_status ,
proc_nice , proc_open , proc_terminate , shell_exec , highlight_file ,
escapeshellcmd , define_syslog_variables , posix_uname , posix_getpwuid ,
apache_child_terminate , posix_kill , posix_mkfifo , posix_setpgid ,
posix_setsid , posix_setuid , escapeshellarg , posix_uname , ftp_exec ,
ftp_connect , ftp_login , ftp_get , ftp_put , ftp_nb_fput , ftp_raw , ftp_rawlist ,
ini_alter , ini_restore , inject_code , syslog , openlog ,
define_syslog_variables , apache_setenv , mysql_pconnect , eval , phpAds_XmlRpc
, phpAds_remoteInfo , phpAds_xmlrpcEncode , phpAds_xmlrpcDecode ,
xmlrpc_entity_decode , fp , fput"
[...]
```

Dass eine Funktion nicht gesperrt werden darf, wenn diese im Code benötigt wird, ist natürlich selbstverständlich.

8 Abblocken

Damit all die restlichen Ports, welche sich in Zukunft öffnen könnten, nicht immer gleich zum Sicherheitsrisiko werden, können diese mit einer Firewall geblockt werden.

8.1 iptables

Die iptables sind standardmässig in jedem Linux-Kernel verbaut, weshalb nichts mehr nachinstalliert werden muss. Damit SSH auf Port 33 und Apache auf Port 443 immer noch verfügbar sind, wird alles blockiert ausser diesen (Achtung: Alle alten Rules werden gelöscht durch die ersten 5 Zeilen):

```
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X
iptables -P FORWARD DROP
iptables -A INPUT -m state --state INVALID -j DROP
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -p icmp -j ACCEPT
iptables -A INPUT -p tcp --dport 33 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -j DROP
```

9 Hilfsmittel

Neben der durchdachten Konfiguration von einzelnen Diensten gibt es noch unzählige Zusatzsoftware, welche den Server um Funktionen und Funktionalitäten erweitern, um das Sicherheitslevel noch weiter zu heben.

9.1 Anklopfen

Anklopfen kennt jedem von zu Hause. Vor verschlossener Türe klopft man an, um zu melden, dass man gerne rein möchte. Von innen wird dann geprüft, ob man berechtigt ist, wodurch sich die Türe öffnet oder eben nicht. Genauso geht das auch in der IT. Das Tool knockd gibt den Zugriff auf gewisse Ports frei, wenn man vorher richtig anklopft. Für einen Dienst wie Apache macht das Anklopfen natürlich nur wenig Sinn, da nicht jeder Benutzer zuerst anklopfen kann, bevor er die Webseite besuchen darf. Bei SSH jedoch, ist es durchaus brauchbar. Dazu muss nach dem Installieren zuerst knockd freigeschaltet werden. Dazu verändert man die Datei `/etc/default/knockd`:

```
START_KNOCKD=1
```

Nun werden in der Datei `/etc/knockd.conf` die Einträge zum Öffnen und Schliessen von SSH definiert:

```
[openSSH]
sequence    = 42,43,44
seq_timeout = 5
command     = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 33 -j
             ACCEPT
tcpflags    = syn

[closeSSH]
sequence    = 44,43,42
seq_timeout = 5
command     = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 33 -j
             ACCEPT
tcpflags    = syn
```

Wird nun der Service gestartet, so kann man über die Kommandozeilenapplikation knock die Ports freigeben oder sperren:

```
knock SERVER 42 43 44
knock SERVER 44 43 42
```

Wichtig dabei zu beachten; man kann auf seinen Server nur zugreifen, wenn vorher die Ports freigegeben wurden. Somit wenn auf einem Computer kein «knock» verfügbar ist, so bekommt man die Ports nicht frei!

9.2 Verbannen

Die Theorie besagt, dass je länger man versucht, man irgendwann das richtige Passwort finden muss. Und genau auf diese Tatsache stützt sich eine Brute-force-Attacke! Hätte ein Angreifer unendlich viel Zeit, so würde er früher oder später das richtige Passwort finden. Damit das aber nicht der Fall ist, sperren wir einen Angreifer von unserem Server aus, wenn er fünf mal das falsche Passwort eingegeben hat. Damit gibt es zwar auch die Chance, dass wir uns selbst aussperren, jedoch in fünf Versuchen, sollte es möglich sein, einmal das richtige Passwort einzutippen. Dazu verwenden wir das Tool fail2ban, welches unseren Server um genau diese Funktionalität erweitert. Da in der Standardkonfiguration schon viel unnötiges Vorkonfiguriert ist, sollte man ab dem Wort «JAILS» alles löschen und neu beginnen. Für SSH fügen wir nun den folgenden Eintrag wieder zu:

```
[ssh]
enabled = true
port    = 33
filter  = sshd
logpath = /var/log/auth.log
maxretry = 5
```

Versucht nun eine automatische Attacke meinen Server zu knacken, so wird diese nach spätestens fünf Versuchen über die iptables gebannt und kann nicht mehr auf meinen Server zugreifen.

Hat man auf seinem Apache noch ein htaccess-Login, so könnte man auch das Schützen:

```
[apache]
enabled = true
port    = http, https
filter  = apache-auth
logpath = /var/log/apache*/error.log
maxretry = 5
```

Oder aber noch viele viele mehr! Am besten man schaut sich vor dem Löschen noch die Beispiele durch.

9.3 Mod Security

Mod_Security ist eine Software Firewall, welche Kontrolliert, welche Eingaben auf einem Apache gmacht und was für Daten übermittelt werden. Nachdem die Software installiert wurde kann die Konfiguration begonnen werden. Diese beschränkt sich auf das Herunterladen der Regeln und dem Aktivieren der Firewall. Unter Debian heisst das benötigte Paket libapache-mod-security. Nach dem Installieren wird eine Datei unter /etc/apache2/conf.d/modsecurity2.conf angelegt und mit folgendem inhalt ergänzt:

```
<ifmodule mod_security2.c>
Include conf.d/modsecurity/*.conf
</ifmodule>
```

Danach werden noch die nötigen Ordner erstellt und verlinkt:

```
sudo mkdir /var/log/apache2/mod_security
sudo ln -s /var/log/apache2/mod_security/ /etc/apache2/logs
```

Und die aktuellsten Regeln heruntergeladen:

```
sudo mkdir /etc/apache2/conf.d/modsecurity
cd /etc/apache2/conf.d/modsecurity
sudo wget http://www.modsecurity.org/download/modsecurity-core-rules_2
.5-1.6.1.tar.gz
sudo tar xzvf modsecurity-core-rules_2.5-1.6.1.tar.gz
```

Nun noch das Modul aktivieren und Apache neustarten:

```
sudo a2enmod mod-security
sudo /etc/init.d/apache2 restart
```

Und von nun an wird der Apache durch Mod_Security geschützt.