Webserver

- Tomcat
  - Tomcatuser in Debian Jessie mit Tomcat 7
- Apache
  - Zugriff auf .git und .svn Unterverzeichnisse verbieten
  - Apache Grundinstallation und Absicherung mit SSL und mod_php
  - schlechte Perfomance
  - Apache mit PHP-FPM
  - Apache mit php5-fpm installieren
  - Authentifizierung via LDAP
  - MultiViews
  - HSTS-Header (HTTP Strict Transport Security) konfigurieren
  - Sichere SSL Konfiguration
  - htaccess Verzeichnis schützen
  - Performancetuning
  - Permanente Umleitung auf HTTPS
  - Reverse Proxy
  - SSL Reverse Proxy
  - TRACK und TRACE ausschalten
  - virtuellen Host konfigurieren
  - X-Frame-Options setzen
  - Eigene Startseite für einen VHost mit vielen Aliasen
  - weitere Möglichkeiten der Weiterleitung (ohne mod_rewrite)
  - Weiterleitung ohne URL-Änderung (mit mod_rewrite)
  - mod_rewrite - /index.php/ aus URL entfernen
- Lighttpd
  - HSTS-Header (HTTP Strict Transport Security) konfigurieren
  - Permanente Umleitung auf HTTPS
- nginx
  - HSTS Header "richtig" setzen
  - HSTS-Header (HTTP Strict Transport Security) konfigurieren
  - Logfiles dynamisch benennen

- nginx page caching
- Permanente Umleitung auf HTTPS
- Query-String an PHP-FPM mit übergeben
- Reverse Proxy
- Reverse Proxy um HTTP Basic Auth zu entfernen
- Rewrite einer Datei auf Datei-$Hostname
- Sichere SSL Konfiguration
- Tuning nginx
- X-Frame-Options setzen
- Zugriff auf .git und .svn Unterverzeichnisse verbieten

Tomcat

Tomcatuser in Debian Jessie mit Tomcat 7

Installierte Pakete:

```
aptitude install tomcat7 tomcat7-admin tomcat7-common libtomcat7-java openjdk-8-jre-headless
openjdk-8-jdk-headless
```

Die tomcat-users.xml dann wie folgt abändern:

/etc/tomat7/tomcat-users.xml

```
<tomcat-users>
  <role rolename="manager-gui"/>
  <role rolename="manager-script"/>
  <role rolename="manager-jmx"/>
  <role rolename="manager-status"/>
  <role rolename="admin-gui"/>
  <role rolename="admin-script"/>
  <user username="admin" password="passwort" roles="manager-gui,manager-script,manager-jmx,
</tomcat-users>
```

Abschließend den Tomcat via `systemctl restart tomcat7.service` neu starten.

Apache

Apache

Zugriff auf .git und .svn Unterverzeichnisse verbieten

Mit folgendem Snippet läßt sich der Zugriff auf .git und .svn Verzeichnisse verhindern. Das funktioniert entweder per Vhost oder auch global, z.B. als Datei in /etc/apache2/conf.d/.

```
# Apache 2.4
<DirectoryMatch ".*/\.(git|svn)/.*">
    Require all denied
</DirectoryMatch>

# Apache 2.2
<DirectoryMatch ".*/\.(git|svn)/.*">
    Order deny,allow
    Deny from all
</Directorymatch>
```

Alternativen per Einzeiler (braucht mod_alias):

```
AliasMatch \.(svn|git) /404-error.html
# oder:
RedirectMatch 404 /\.(git|svn)
```

Apache Grundinstallation und Absicherung mit SSL und mod_php

> Diese Anleitung wurde für Debian Jessie geschrieben.

In diesem Artikel geht es darum Apache mit PHP (mod_php) zu installieren und grundlegend sicher zu konfigurieren.

Folgendes soll erreicht werden (Stand 11.12.2015):

- Keine Ausgabe von Versionsinformationen bei Apache und PHP
- Verwendung sicherer Ciphers und Hashingalgorithmen bei SSL
- Versteckte Dateien sollen nicht abrufbar sein (z.B. .git- oder .svn-Verzeichnisse)
- DocumentRoots sicher machen

Nach Änderungen an den Konfigurationsdateien die Apache-Syntax testen und neu laden:

```
# apache2ctl configtest
Syntax OK
# systemctl restart apache2.service
```

Installation von apache2.4 mit PHP und SSL

```
aptitude install apache2 libapache2-mod-php5 php5
```

Apache 2.4 absichern

Ausgabe von Versionsinformationen bei Apache deaktivieren

/etc/apache2/conf-available/security.conf

```
ServerTokens Prod
ServerSignature Off
```

ggf. zusätzliche Header mitsenden (mod_headers aktivieren: „a2enmod headers“)

/etc/apache2/conf-available/security.conf

```
# prevent MSIE from interpreting files as something else than declared by HTTP headers
Header set X-Content-Type-Options: "nosniff"
# disable loading pages from this site as frames
```

```
Header set X-Frame-Options: "sameorigin"
```

Zugriff auf Verzeichnisse von Versionskontrollsystemen wie Git und SVN unterbinden. Aktivieren mit „a2enconf cvs-deny“

/etc/apache2/conf-available/cvs-deny.conf

```
RedirectMatch 403 (?i)\.git
RedirectMatch 403 (?i)\.svn
```

ggf. Zugriff auf alle versteckten Verzeichnisse unterbinden (kann auch Probleme machen). Aktivieren mit „a2enconf hidden-deny“

/etc/apache2/conf-available/hidden-deny.conf

```
RedirectMatch 404 (?i)/\..+
```

SSL soll möglichst sicher konfiguriert (soweit die alten SSL-Libs bei Debian es erlauben) werden. Wie man SSL mit modernen Ciphers sicher konfiguriert steht in diesem Artikel.

Ggf. mod_security installieren, ein Filter und IDS gegen verschiedene Angriffe auf den Webserver (XSS, SQL injection):

```
# aptitude install libapache2-mod-security2
# cp /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf
```

ein sicherer Apache default-Vhost (auch als Vorlage für weitere Vhosts verwendbar)

/etc/apache2/sites-available/000-default.conf

```
<VirtualHost *:80>
    ServerName www.FQDN
    ServerAlias FQDN
    ServerAdmin webmaster@FQDN
    DocumentRoot /var/www/FQDN

    LogLevel info
    ErrorLog ${APACHE_LOG_DIR}/FQDN-error.log
    CustomLog ${APACHE_LOG_DIR}/FQDN-access.log combined
</VirtualHost>

<VirtualHost _default_:443>
    ServerName www.FQDN
    ServerAlias FQDN
    ServerAdmin webmaster@FQDN
```

```
    DocumentRoot /var/www/FQDN

    LogLevel info ssl:warn
    ErrorLog ${APACHE_LOG_DIR}/FQDN-error.log
    CustomLog ${APACHE_LOG_DIR}/FQDN-access.log combined

    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/FQDN.pem    # contains the domain certificate and
intermediate certificates
    SSLCertificateKeyFile /etc/apache2/ssl/FQDN.key
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

PHP absichern

Ausgabe von Versionsinformationen bei PHP deaktivieren

/etc/php5/apache2/php.ini

```
expose_php=Off
```

bei Bedarf können weitere Sicherungsmaßnahmen getroffen werden

/etc/php5/apache2/php.ini

```
# ggf. Dateiuploads deaktivieren
file_uploads=Off

# Ausführung von entferntem Code unterbinden
allow_url_fopen=Off
allow_url_include=Off

# Zugriff auf das Dateisystem einschränken
open_basedir="/var/www/html/"
```

Apache

schlechte Perfomance

folgender Eintrag im error_log

```
child pid 18609 exit signal File size limit exceeded (25)
```

deutet darauf hin, dass der Apache nicht in seine Logfiles schreiben darf, da die max. Größe überschritten wurde. Tritt eigentlich nur bei älteren Systemen auf, die eine Begrenzung auf 2GB haben. Diese Files einfach rotieren/löschen und ein SIGHUP auf den Apache durchführen.

Apache mit PHP-FPM

> Diese Anleitung wurde für Debian Stretch geschrieben

Die Installation von Apache und PHP mit PHP-FPM gestaltet sich inzwischen recht einfach. Zuerst werden alle nötigen Pakete installiert (der performante event-Worker von Apache kommt automatisch mit):

```
aptitude update
aptitude install apache2 php7.0 php7.0-cgi php7.0-cli php7.0-fpm php7.0-mbstring php7.0-mcrypt
php7.0-opcache php7.0-intl php7.0-gd php-apcu php-apcu-bc ssl-cert
```

proxy_fcgi und die PHP-FPM Konfiguration aktivieren:

```
a2enmod proxy_fcgi
a2enconf php7.0-fpm
systemctl restart apache2.service
```

PHP Installation testen:

```
echo "<?php phpinfo();" > /var/www/html/index.php
```

Unter http:*<servername>/index.php sollte nun die PHP Infoseite zu sehen sein.*

Apache mit php5-fpm installieren

> Diese Anleitung wurde für Debian Wheezy geschrieben.

Voraussetzung: Frisch installierter Debian-Standardserver.

```
apt-get update
apt-get -y install apache2 apache2-mpm-worker apache2-utils php5 php5-fpm php5-common php5-gd
php5-mysql php5-imap php5-cli libapache2-mod-fastcgi php-pear php-auth php5-mcrypt mcrypt
php5-curl php5-intl php5-ming php5-ps php5-pspell php5-recode php5-snmp php5-sqlite php5-
xmlrpc php5-xsl
```

Jetzt wird Apache noch mit php5-fpm verheiratet. Dazu das folgende File nach /etc/apache2/conf.d/ legen:

/etc/apache2/conf.d/php5-fpm

```
<FilesMatch ".+\.ph(p[345]?|t|tml)$">
    SetHandler application/x-httpd-php
</FilesMatch>
<FilesMatch ".+\.phps$">
    SetHandler application/x-httpd-php-source
    Order Deny,Allow
    Deny from all
</FilesMatch>
<FilesMatch "^\.ph(p[345]?|t|tml|ps)$">
    Order Deny,Allow
    Deny from all
</FilesMatch>
Action application/x-httpd-php /fcgi-bin/php5-fpm virtual
Alias /fcgi-bin/php5-fpm /fcgi-bin-php5-fpm
<Location /fcgi-bin/php5-fpm>
  Order Deny,Allow
  Deny from All
  Allow from env=REDIRECT_STATUS
</Location>
FastCgiExternalServer /fcgi-bin-php5-fpm -socket /var/run/php5-fpm.sock -pass-header
Authorization -idle-timeout 3600 -flush
```

Benötigte Apache-Module aktivieren und neu starten:

```
a2enmod actions fastcgi
service apache2 restart
```

Ein Testfile für PHP anlegen:

/var/www/test.php

```
<?php phpinfo(); ?>
```

Abschließend noch den Apache neu starten (service apache2 restart) und das Testfile im Browser aufrufen (http://dein-server.de/test.php). Die PHP-Infoseite sollte nun angezeigt werden.

Die php.ini liegt bei dieser Konfiguration in /etc/php5/fpm/php.ini. Verwendete Module dort im conf.d-Unterverzeichnis. PHP-FPM selbst wird dort in php-fpm.conf konfiguriert. Standardmäßig gibts nur einen Pool, der mit dem User www-data läuft. Dieser Pool wird in /etc/php5/fpm/pool.d/www.conf konfiguriert. In dieser Datei kann auch das Performance-Tuning von FPM erfolgen.

Authentifizierung via LDAP

Getestet unter CentOS-5 32bit mit Apache 2.2.3

- Installation der nötigen Module:

```
yum -y install mod_authz_ldap
```

- Die Änderung kann in jeder Vhost-, Directory-, Location-Direktive oder z.B. gleich in der Datei /etc/httpd/conf.d/authz_ldap.conf vorgenommen werden. Der folgende Eintrag schützt z.B. den ganzen Auftritt:

```
<Location />
        AuthType Basic
        AuthName "LDAP-Authentifizierung"
        AuthBasicProvider ldap
        AuthzLDAPAuthoritative off
        AuthLDAPURL
"ldap://mein.ldapserver.de/ou=People,dc=ldapserver,dc=de?uid?sub?(objectClass=*)"
        require valid-user
</Location>
```

SSL-verschlüsselte Kommunikation zum Webserver aktivieren

um den LDAP-Server verschlüsselt anzusprechen (wenn sich z.B. Web- und LDAP-Server in verschiedenen Hosting-Locations befinden, würde keiner machen, aber nur mal als Beispiel) sind 2 kleine Änderungen notwendig.

- Check des Serverzertifikates deaktivieren (oder das CA-Cert auf jedem Webserver einbinden). Dazu folgende Zeile am besten in die Datei /etc/httpd/conf.d/authz_ldap.conf eintragen:

```
LDAPVerifyServerCert off
```

- die AuthLDAPURL auf „ldaps:*…“ abändern*

Apache

MultiViews

oder „Warum liefert Apache etwas aus, dass es gar nicht gibt?“

MultiViews gehören zu mod_negotiation und ermöglichen dem Apache Dateinamen zu „erraten“.

Bekommt der Apache bei aktivierten MultiViews einen Request auf eine Datei, die es gar nicht gibt, z.B. /var/www/bla, aber es gibt eine Datei namens /var/www/bla.png, dann wird diese .png Datei ausgeliefert (die URL ändert sich trotzdem nicht). Apache sucht nach bla.* und liefert den besten Treffer (mit allen dazugehörigen Media-Typen, content encodings und Headern) aus.

Leider hat das auch negative Folgen. Wenn es z.B. einen Rewrite auf /bla gibt, funktioniert dieser nicht! Also lieber alles sauber verlinken oder programmieren und auf MultiViews verzichten, als sich hinterher über dieses Verhalten zu ärgern.

MultiViews sind standardmäßig aktiviert (/etc/apache2/sites-enabled/default), zum Deaktivieren einfach den Eintrag entfernen oder explizit ein - vorne ran setzen:

```
...
<Directory /var/www>
Options Indexes FollowSymLinks -MultiViews
...
```

Apache

HSTS-Header (HTTP Strict Transport Security) konfigurieren

HTTP Strict Transport Security (abgekürzt HSTS, definiert in RFC6797) ist ein Sicherheitsfeature einer Webseite, das dem Besucher, bzw. dessen Browser sagt, dass sie nur noch per HTTPS verschlüsselt mit ihm kommunizieren will. Dazu wird ein zusätzlicher HTTP-Header gesetzt, der Angaben zum Zeitraum, Umgang mit Subdomains und der Verwendung der HSTS Preloadliste enthält.

Das Feature funktioniert folgendermaßen: Ein Besucher tippt z.B. www.seite-x.de in seinen Browser ein. Der Webserver leitet ihn auf HTTPS um. In der HTTPS-Verbindung wird ein zusätzlicher HTTP-Header gesendet, der bestimmte Informationen für den Browser enthält. Der Browser merkt sich das für den angegebenen Zeitraum. Bei zukünftigen Besuchen greift der Browser dann sofort auf die HTTPS Seite zu.

Es gibt folgende Keywörter:

- `max-age=63072000;` ? innerhalb dieses Zeitraums wird direkt HTTPS angesteuert (Angabe in Sekunden).
- `includeSubDomains;` ? der Eintrag gilt auch für sämtliche anderen Subdomains (vorsichtig damit, wenn andere Subdomains z.B. nicht per HTTPS funktionieren).
- `preload;` ? Google pflegt eine Liste mit Webseiten, die HSTS aktiviert haben. Diese Liste ist in aktuellen Versionen von Chrome, Firefox, Safari, IE11 und Edge enthalten. Diese Webseiten werden sofort per HTTPS angesurft. In die Liste kann man seine eigenen Webseiten jederzeit selbst eintragen, sofern die Voraussetzungen erfüllt werden.

Beim Apache-Webserver muss dazu folgender Eintrag in dem entsprechenden HTTPS-Vhost hinterlegt werden (vorher „`a2enmod headers`“ ausführen):

```
Header always set Strict-Transport-Security "max-age=63072000; includeSubDomains; preload"
```

Die HTTP-Version der Webseite muss nun permanent auf HTTPS umgeleitet werden. Wie das für Apache einzurichten ist, habe ich hier beschrieben.

Danach muss Apache neu geladen/neu gestartet werden!

Wie man SSL mit modernen Ciphers und sicher konfiguriert steht in diesem Artikel.

Apache

Sichere SSL Konfiguration

Mit dem „Mozilla SSL Configuration Generator“ läßt sich aktuell wohl am einfachsten und schnellsten eine sichere Konfiguration für den nginx-Webserver erstellen.

Einfach Webserver, Version von Webserver und OpenSSL setzen und zwischen verschiedenen Cipherprofilen wählen, fertig ist eine Beispielkonfiguration.

Der Eintrag zu TLS im Mozilla Wiki ist auch sehr lesenswert.

Alternativ habe ich hier noch meine (**veraltete**) Anleitung:

Die Empfehlungen richten sich nach den Vorgaben von BetterCrypto.org und ihrem Applied Crypto Hardening PDF mit dem Stand vom 21.04.2016.

Folgendes kann man machen, um die Standard-SSL Konfiguration von Apache 2.4 sicherer zu gestalten:

- Schwache Cipher und Protokolle abschalten
- Webserver soll die Reihenfolge der Ciphers vorgeben
- SSL-Komprimierung ausschalten
- HSTS aktivieren
- Permanente Umleitung von HTTP nach HTTPS aktivieren

Das folgende gilt für Apache 2.4 unter Debian. Ich trage dabei nur die beiden Direktiven für SSL-Zertifikatsbundle (SSLCertificateFile) und SSL-Keyfile (SSLCertificateKeyFile) direkt in den jeweiligen Vhost ein. Die restliche SSL-Konfiguration setze ich global (nach „`a2enmod ssl`“) in /etc/apache2/mods-enabled/ssl.conf. Vorher die bereits konfigurierten Settings auskommentieren (es sollen keine Direktiven doppelt eingetragen sein) und diesen Teil am Ende der Datei einfügen:

```
SSLProtocol All -SSLv2 -SSLv3
SSLHonorCipherOrder On
SSLCompression off
SSLCipherSuite 'EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+SHA256:EECDH:+CAMELLIA128:+AE
```

Danach muss Apache neu geladen/neu gestartet werden!

Apache

htaccess Verzeichnis schützen

Einfachstes Beispiel - ein Verzeichnis mit Username/Passwort-Abfrage schützen:

folgendes im zu schützenden Verzeichnis in der Datei .htaccess einfügen:

```
AuthType Basic
AuthName "Interner Bereich"
AuthUserFile /var/www/html/.htpasswd
Require valid-user
```

damit dürfen sich alle in der .htpasswd existierenden Benutzer anmelden.

neue .htpasswd im aktuellen Verzeichnis erzeugen:

```
htpasswd -c .htpasswd <username>
```

einen neuen Benutzer zu vorhandener .htpasswd hinzufügen (mit diesem Befehl lassen sich auch Kennwörter ändern):

```
htpasswd .htpasswd <username>
```

weiterführende Infos

Performancetuning

Folgende Fehlermeldung taucht im Log auf:

```
[Fri Jul 13 10:48:59 2007] [error] server reached MaxClients setting, consider raising the
MaxClients setting
```

Folgende Parameter beeinflussen die Server-Performance enorm. Grundsätzlich kann man mit folgenden Parametern mit dem Performance-Tuning beginnen:

```
Timeout 15

KeepAlive On
MaxKeepAliveRequests 1000
KeepAliveTimeout 2

<IfModule prefork.c>
StartServers      10
MinSpareServers   10
MaxSpareServers   20
ServerLimit      250
MaxClients       250
MaxRequestsPerChild 8000
</IfModule>
```

Apache

Permanente Umleitung auf HTTPS

Um Besucher permanent auf HTTPS umzuleiten kann dieser einfache Codeteil im HTTP-Vhost (Port 80) eingetragen werden:

```
<VirtualHost *:80>
  ServerName seite-x.de
  Redirect permanent / https://seite-x.de/
  [...]
</VirtualHost>
```

Alternativ geht das auch mit mod_rewrite (vorher „`a2enmod rewrite`“ ausführen):

```
<VirtualHost *:80>
  ServerName seite-x.de
  RewriteEngine On
  RewriteCond %{HTTPS} off
  RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI}
  [...]
</VirtualHost>
```

Umleitung auf https nur für bestimmte Domains:

```
# seite-z.de soll NICHT umgeleitet werden
<VirtualHost *:80>
  ServerName seite-x.de www.seite-x.de
  ServerAlias seite-y.de www.seite-y.de
  ServerAlias seite-z.de www.seite-z.de

  RewriteEngine On
  RewriteCond %{HTTPS} off
  RewriteCond %{HTTP_HOST} ^(www\.)seite-x.de$ [OR]
  RewriteCond %{HTTP_HOST} ^(www\.)seite-y.de$
  RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI} [R=301,L]

  # alternativ gleiche Regel mit seite-z.de negiert in der RewriteCond
```

Danach muss einmal der Apache neu gestartet werden. Wie man SSL mit modernen Ciphers und sicher konfiguriert steht in diesem Artikel.

Reverse Proxy

Reverse Proxies werden (neben der Möglichkeit eine SSL-Version einer Webseite einfach einzubinden) auch benutzt, um z.B. eine Applikation, die auf einem anderen Port läuft, über Port 80 verfügbar zu machen. Typisches Beispiel wäre eine Java-App, die auf Port 8080 läuft.

- benötigte Module in der Apache-Config aktivieren, unter Debian geht das einfach mit „a2enmod proxy proxy_http“ oder in anderen Distris diese beiden Zeilen in der httpd.conf einkommentieren:

```
LoadModule  proxy_module         modules/mod_proxy.so
LoadModule  proxy_http_module    modules/mod_proxy_http.so
```

- jetzt folgendes in der VHost-Konfiguration eintragen (ACHTUNG: in diesem Beispiel wird der externe Inhalt direkt unter / eingebunden):

```
ProxyRequests off
ProxyPreserveHost on
ProxyPass / http://localhost:8080/
ProxyPassReverse / http://localhost:8080/
```

- soll es über ein Unterverzeichnis erreichbar sein, wird der Code etwa so angepasst:

```
ProxyRequests off
ProxyPreserveHost on
ProxyPass /applikation http://localhost:8080/
ProxyPassReverse /applikation http://localhost:8080/
```

- jetzt noch „service apache2 reload“, danach kann die Applikation z.B. als http://meinserver.de/applikation1 aufgerufen werden.

Apache

SSL Reverse Proxy

Ein SSL Reverse Proxy ermöglicht es eine bestimmte Webseite über ein bereits vorhandenes Zertifikat über SSL abrufbar zu machen. Beispiel: Ihr habt eine Domain www.kundenshops.de gekauft und für diese ein SSL-Zertifikat gekauft. Jetzt haben Eure Kunden eigene Webshops aufgemacht und fragen Euch, wie sie diese günstig absichern können, damit z.B. Kundendaten verschlüsselt übertragen werden.

Statt jetzt jedem Kunden ein teures SSL-Zertifikat und eine eigene IP-Adresse zu verkaufen, könnt ihr seinen Shop auch per SSL Reverse Proxy absichern. Das bedeutet, dass der Shop des Kunden unter Eurem Zertifikat aufrufbar sein wird, eine URL würde dann z.B. so aussehen: https://www.kundenshops.de/www.mein-autoshop.de oder https://www.kundenshops.de/www.nasenfloetenshop.de.

Wie konfiguriere ich nun meinen Apache, damit das so funktioniert?

- benötigte Module in der Apache-Config aktivieren, unter Debian geht das einfach mit „a2enmod proxy proxy_http“ oder in anderen Distris diese beiden Zeilen in der httpd.conf einkommentieren:

```
LoadModule  proxy_module         modules/mod_proxy.so
LoadModule  proxy_http_module    modules/mod_proxy_http.so
```

- nun muss in der Konfiguration des SSL-Vhosts www.kundenshops.de folgendes eingetragen werden:

```
RewriteRule ^/www.nasenfloetenshop.de$ /www.nasenfloetenshop.de/ [R,L]
ProxyPass /www.nasenfloetenshop.de/ http://www.nasenfloetenshop.de/
<Location /www.nasenfloetenshop.de/>
  ProxyPassReverse /
</Location>
```

- nach einem „service httpd reload“ steht nun der Inhalt von www.nasenfloetenshop.de SSL gesichert unter der URL https://www.kundenshops.de/www.nasenfloetenshop.de zur Verfügung.

> Bei manchen CMS, wie z.B. Wordpress, sind noch Anpassungen in der Konfiguration nötig, da etwa die SITE_URL fest hinterlegt ist und dynamisiert werden muss. Weitere Infos dazu finden sich zuhauf im Netz.

Wie man SSL mit modernen Ciphers und sicher konfiguriert steht in diesem Artikel.

Apache

TRACK und TRACE ausschalten

von Heise-Security

HTTP-Methoden ermöglichen unerlaubtes Ausspähen von Cookies

Ist in einem Cookie die Option httpOnly definiert, darf nur der entsprechende Server darauf zugreifen. Versucht ein Skript den Inhalt des Cookies über die Methode document.cookie abzufragen, wird der Inhalt nicht zurückgeliefert. Damit lassen sich so genannte Cross-Site-Scripting-Attacken (XSS) abwehren. Das CERT/CC weist nun aber in einer Vulnerability Note darauf hin, dass Microsofts Internet Information Server (IIS) eine HTTP-Methode unterstützt, mit der sich solche Cookies trotzdem heimlich ausspähen lassen. Dieses Problem lässt sich allerdings auch analog auf andere Server wie den Apache übertragen.

Zur Fehlersuche ist im HTTP-Protokoll die TRACE-Methode definiert, die als Antwort die ursprüngliche Anfrage eines Web-Clients zurücksendet. Erfordert der Webserver eine Authentifizierung per Cookie, sendet der Client diese mit. In der HTTP-Antwort ist dementsprechend auch das Cookie enthalten. Mit einem Skript kann man nun das Cookie auslesen und anzeigen.

Um nicht das Sicherheitszonen-Modell zu verletzen, darf ein Skript nur mit der Domäne kommunizieren, aus der es stammt. Diverse Fehler in Web-Browsern ermöglichen es aber, Scripting-Code auch in anderen Domänen auszuführen. Damit kann ein Angreifer über eine manipulierte Webseite Code auf einem Client ausführen und über die TRACE-Methode andere Server ansprechen und Cookies auslesen. Diese Art von Angriff bezeichnet man als Cross-Site-Tracing-Angriffe (XST).

Die TRACE-Methode sollte aus Sicherheitsgründen auf Servern im Internet deaktiviert sein, die Anfrage eines Clients bleibt dann ohne Antwort. Ist sie doch aktiviert, schreibt der Server solche Anfragen immerhin in seine Log-Files. Microsoft hat im IIS eine eigene TRACE-Methode implementiert: TRACK. Diese Methode ist zwar nicht vollständig dokumentiert, erfüllt aber weitestgehend den gleichen Zweck wie TRACE. Allerdings loggt der IIS derlei Anfragen nicht mit. Damit kann man einen Angriff nicht mehr nachvollziehen.

Um beim Apache-Webserver die TRACE-Methode zu deaktivieren ist folgende Rewrite-Rule erforderlich:

```
# TRACK und TRACE deaktivieren - http://www.heise.de/security/news/meldung/43354
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
RewriteRule .* - [F,L]
```

Beim IIS lassen sich TRACE- und TRACK-Anfragen mit dem Tool URLScan ausfiltern. Als Methoden sollten nur noch GET, HEAD und POST erlaubt sein.

Apache

virtuellen Host konfigurieren

Beispiel fuer einen NamedVirtualHost (HTTP + HTTPS):

```
NameVirtualHost *:80
NameVirtualHost *:443

<VirtualHost _default_:80>
   ServerAdmin     webmaster@example.com
   ServerName      www.example.com
   ServerAlias     example.com
   DocumentRoot    /var/www/html/example.com
   ScriptAlias     /cgi-bin/ /var/www/html/example.com/cgi-bin/

   ErrorLog        logs/example.com-error_log
   CustomLog       logs/example.com-access_log combined

   RewriteEngine On

   # Umleitung auf komplette URL www.example.com
   RewriteCond %{HTTP_HOST}   !^www\.example\.com [NC]
   RewriteCond %{HTTP_HOST}   !^$
   RewriteRule ^/(.*)         http://www.example.com/$1 [L,R]

   # TRACK und TRACE deaktivieren - http://www.heise.de/security/news/meldung/43354
   RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
   RewriteRule .* - [F,L]
</VirtualHost>

<VirtualHost _default_:443>
   ServerAdmin     webmaster@example.com
   ServerName      www.example.com
   ServerAlias     example.com
   DocumentRoot    /var/www/html/example.com
   ScriptAlias     /cgi-bin/ /var/www/html/example.com/cgi-bin/

   ErrorLog logs/example.com-error_log
```

```
   TransferLog logs/example.com-access_log
   CustomLog logs/example.com-ssl_request_log "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
   LogLevel warn

   SSLEngine on
   #SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP
   SSLCipherSuite SSLv3
   SSLCertificateFile /etc/pki/tls/certs/www.example.com.crt
   SSLCertificateKeyFile /etc/pki/tls/private/www.example.com.key
   SSLCACertificateFile /etc/pki/tls/certs/cacert.pem

   <Files ~ "\.(cgi|shtml|phtml|php3?)$">
     SSLOptions +StdEnvVars
   </Files>

   <Directory "/var/www/html/example.com/cgi-bin">
     SSLOptions +StdEnvVars
   </Directory>

   SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown downgrade-1.0 force-response-1.0

   RewriteEngine On

   RewriteCond %{HTTP_HOST}   !^www\.example\.com [NC]
   RewriteCond %{HTTP_HOST}   !^$
   RewriteRule ^/(.*)         https://www.example.com/$1 [L,R]

   RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
   RewriteRule .* - [F,L]
</VirtualHost>
```

Apache

X-Frame-Options setzen

Über den X-Frame-Options Header (RFC 7034) kann eine Webseite den Browser anweisen, dass sie nicht (ggf. entfernt) über <frame> oder <iframe> geladen werden darf. Das soll sog. Clickjacking verhindern.

Im Apache läßt sich das so setzen (vorher „`a2enmod headers`“):

```
Header always set X-Frame-Options DENY
```

Danach den Apache reloaden/neu starten.

Folgende Optionen sind möglich:

- DENY ? Es ist keinerlei Einbettung per Frame erlaubt, egal welche Seite es versucht.
- SAMEORIGIN ? Es ist nur die Einbettung in Seiten vom gleichen Ursprung erlaubt.
- ALLOW-FROM ? Es ist nur die Einbettung in Seiten vom angebenen Ursprung erlaubt.

> aktuell wird ALLOW-FROM nur von Firefox und Internet Explorer / Edge unterstützt.

Apache

Eigene Startseite für einen VHost mit vielen Aliasen

Problem: Wir haben einen VHost mit vielen Aliasen und jeder konfigurierte Alias soll eine eigene Startseite bekommen. ServerName ist z.B. domain.de, die Aliase tolledomain.de und ganzanderedomain.de sind im gleichen VHost konfiguriert, haben also auch das gleiche DocumentRoot. Ruft man jetzt die URLs auf, würde man auf der index.html landen. Um jetzt jedem VHost eine eigene Startseite zu geben, legt man im DocumentRoot eine .htaccess-Datei mit folgendem Inhalt an:

```
RewriteEngine On
RewriteCond     %{HTTP_HOST}    ^(www\.)?tolledomain.de$
RewriteCond     %{REQUEST_URI}  ^/$
RewriteRule     ^(.*)$          /index_tolledomain.html

RewriteCond     %{HTTP_HOST}    ^(www\.)?ganzanderedomain.de$
RewriteCond     %{REQUEST_URI}  ^/$
RewriteRule     ^(.*)$          /index_ganzanderedomain.html
```

Die erste Zeile aktiviert die RewriteEngine. Die nächste Zeile bestimmt die Bedingung für welche URLs die Regel gilt. Durch das „(www\.)?“ gilt diese Bedingung sowohl für tolledomain.de als auch für www.tolledomain.de. Die letzte Zeile lenkt schließlich alle Zugriffe auf /, d.h. alle Zugriffe, die nicht direkt auf eine Datei gehen auf die spezielle index-Seite für diese Domain um.

weitere Möglichkeiten der Weiterleitung (ohne mod_rewrite)

Serverseitige Weiterleitung

- mit PHP:

```
<?php
header("Status: 301 Moved Permanently");
header("Location:http://www.bla.de");
exit;
?>
```

- .htaccess-Weiterleitung, im DocumentRoot oder im gewünschten Unterordner des Auftritts eine Datei .htaccess mit folgendem Inhalt anlegen:

```
Redirect permanent / http://www.bla.de
oder
Redirect permanent /dir/bla.html http://www.bla.de/bla/bla.html
```

clientseitige Weiterleitung

- durch ein Frameset (fremde Seite in eigener HTML-Seite laden, ist unschön und ist nicht gern gesehen):

```
<html>
<head>
<title>sieht eh keiner</title>
</head>
<frameset rows="*">
<frame src="http://www.bla.de" name="Content">
</frameset>
</html>
```

- Metatag im Header einer Datei:

```
<meta http-equiv="refresh" content="0; url=http://www.bla.de">
```

- JavaScript:

```
<script language ="JavaScript">
<!--
window.location.replace('http://www.neueadresse.de');
// -->
</script>
```

oder

```
<script LANGUAGE="JavaScript">
<!--
top.location.href='http://www.bla.de/'
// -->
</script>
```

Apache

Weiterleitung ohne URL-Änderung (mit mod_rewrite)

Einfach den folgenden Code in der httpd.conf oder im entsprechenden vhost eintragen:

```
RewriteEngine On
RewriteRule ^/(.*)$ https://www.zielserver.de/$1 [R,NC,P]
```

Apache

mod_rewrite - /index.php/ aus URL entfernen

für lesbare und SEO-freundliche URLs kann man die index.php aus der URL entfernen. Das kann z.B. so aussehen:

```
RewriteCond %{THE_REQUEST} /index\.php/(.+)\sHTTP [NC]
RewriteRule ^ /%1 [NE,L,R]
```

außerdem müssen dann Zugriffe auf nicht-existente Dateien auf die index.php umgeleitet werden. Die Clients bekommen davon nichts mit:

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ /index.php/$1 [L]
```

Damit das funktioniert muss natürlich das rewrite-Modul aktiviert sein (`a2enmod rewrite`).

Lighttpd

HSTS-Header (HTTP Strict Transport Security) konfigurieren

HTTP Strict Transport Security (abgekürzt HSTS, definiert in RFC6797) ist ein Sicherheitsfeature einer Webseite, das dem Besucher, bzw. dessen Browser sagt, dass sie nur noch per HTTPS verschlüsselt mit ihm kommunizieren will. Dazu wird ein zusätzlicher HTTP-Header gesetzt, der Angaben zum Zeitraum, Umgang mit Subdomains und der Verwendung der HSTS Preloadliste enthält.

Das Feature funktioniert folgendermaßen: Ein Besucher tippt z.B. www.seite-x.de in seinen Browser ein. Der Webserver leitet ihn auf HTTPS um. In der HTTPS-Verbindung wird ein zusätzlicher HTTP-Header gesendet, der bestimmte Informationen für den Browser enthält. Der Browser merkt sich das für den angegebenen Zeitraum. Bei zukünftigen Besuchen greift der Browser dann sofort auf die HTTPS Seite zu.

Es gibt folgende Keywörter:

- `max-age=63072000;` ? innerhalb dieses Zeitraums wird direkt HTTPS angesteuert (Angabe in Sekunden).
- `includeSubDomains;` ? der Eintrag gilt auch für sämtliche anderen Subdomains (vorsichtig damit, wenn andere Subdomains z.B. nicht per HTTPS funktionieren).
- `preload;` ? Google pflegt eine Liste mit Webseiten, die HSTS aktiviert haben. Diese Liste ist in aktuellen Versionen von Chrome, Firefox, Safari, IE11 und Edge enthalten. Diese Webseiten werden sofort per HTTPS angesurft. In die Liste kann man seine eigenen Webseiten jederzeit selbst eintragen, sofern die Voraussetzungen erfüllt werden.

Beim Lighttpd muss dazu folgender Eintrag in der Konfiguration hinterlegt werden:

/etc/lighttpd/lighttpd.conf

```
server.modules += ( "mod_setenv" )
$HTTP["scheme"] == "https" {
  setenv.add-response-header  = ( "Strict-Transport-Security" => "max-age=63072000;
includeSubDomains; preload")
}
```

Nicht vergessen im Vhost für Port 80 die permanente Umleitung auf HTTPS zu konfigurieren. Wie das für Lighttpd zu machen ist, habe ich hier beschrieben.

Danach muss Lighttpd neu geladen/neu gestartet werden!

Permanente Umleitung auf HTTPS

Um einen bestimmten Vhost auf HTTPS umzuleiten nutze ich dieses Beispiel:

```
$HTTP["scheme"] == "http" {
  $HTTP["host"] == "seite-x.de" {
    url.redirect = (".*" => "https://seite-x.de$0")
  }
}
```

Oder nur eine bestimmte URL auf HTTPS umleiten (bei allen Hosts):

```
$HTTP["scheme"] == "http" {
  $HTTP["host"] == "seite-x.de" {
    url.redirect = ("^/phpmyadmin/.*" => "https://seite-x.de$0")
  }
}
```

Oder um alle konfigurierten Seiten auf HTTPS umzuleiten:

```
$HTTP["scheme"] == "http" {
  $HTTP["host"] =~ ".*" {
    url.redirect = (".*" => "https://%0$0")
  }
}
```

nginx

nginx

HSTS Header "richtig" setzen

Viele nginx-User haben HTTP und HTTPS in einem server-Block zusammengefasst. Trägt man dort nun den add_header Code ein, wird er auch für beide Protokolle ausgeliefert:

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload";
```

Das ist eigentlich falsch, da laut RFC 6797 der HSTS Header nicht über unverschlüsseltes HTTP gesendet werden soll.

Auf jeden Fall RFC konform ist diese Lösung und es lassen sich damit auch Port 80 und 443 zusammenfassen:

```
map $scheme $hsts_header {
    https "max-age=31536000; includeSubDomains; preload";
}

server {
    listen  80;
    listen  443 ssl;

    add_header Strict-Transport-Security $hsts_header;
}
```

nginx

HSTS-Header (HTTP Strict Transport Security) konfigurieren

HTTP Strict Transport Security (abgekürzt HSTS, definiert in RFC6797) ist ein Sicherheitsfeature einer Webseite, das dem Besucher, bzw. dessen Browser sagt, dass sie nur noch per HTTPS verschlüsselt mit ihm kommunizieren will. Dazu wird ein zusätzlicher HTTP-Header gesetzt, der Angaben zum Zeitraum, Umgang mit Subdomains und der Verwendung der HSTS Preloadliste enthält.

Das Feature funktioniert folgendermaßen: Ein Besucher tippt z.B. www.seite-x.de in seinen Browser ein. Der Webserver leitet ihn auf HTTPS um. In der HTTPS-Verbindung wird ein zusätzlicher HTTP-Header gesendet, der bestimmte Informationen für den Browser enthält. Der Browser merkt sich das für den angegebenen Zeitraum. Bei zukünftigen Besuchen greift der Browser dann sofort auf die HTTPS Seite zu.

Es gibt folgende Keywörter:

- `max-age=63072000;` ? innerhalb dieses Zeitraums wird direkt HTTPS angesteuert (Angabe in Sekunden).
- `includeSubDomains;` ? der Eintrag gilt auch für sämtliche anderen Subdomains (vorsichtig damit, wenn andere Subdomains z.B. nicht per HTTPS funktionieren).
- `preload;` ? Google pflegt eine Liste mit Webseiten, die HSTS aktiviert haben. Diese Liste ist in aktuellen Versionen von Chrome, Firefox, Safari, IE11 und Edge enthalten. Diese Webseiten werden sofort per HTTPS angesurft.

Bei nginx muss dazu folgender Eintrag im HTTPS-Teil der Konfiguration gesetzt werden:

```
add_header Strict-Transport-Security "max-age=63072000; includeSubDomains; preload";
```

Nicht vergessen im Vhost für Port 80 die permanente Umleitung auf HTTPS zu konfigurieren. Wie das für nginx zu machen ist, habe ich hier beschrieben.

Danach muss nginx neu geladen/neu gestartet werden!

Wer die Behandlung von HTTP und HTTPS Traffic bei nginx in einem server-Block zusammenfassen möchte, sollte diesen Eintrag von mir beachten.

nginx

Logfiles dynamisch benennen

Diese Anleitung wurde unter Debian Squeeze mit nginx 1.2.1 aus den Squeeze-Backports getestet. Der orginal nginx aus Squeeze ist dafür zu alt.

nginx bietet die Möglichkeit Logfiles dynamisch anzulegen und durch die Verwendung von variablen dynamisch zu benennen.

dazu braucht der nginx-worker, der im Gegensatz zum nginx-master nicht als root sondern als www-data läuft, Schreibrechte im Logverzeichnis:

```
chown -R www-data.root /var/log/nginx
```

Außerdem muss das Verzeichnis /etc/nginx/html angelegt werden, da nginx auf dessen Existenz prüft. Warum ist mir unklar.

```
mkdir /etc/nginx/html
```

Jetzt kann die access_log Direktive (z.B. direkt in der nginx.conf) wie gewünscht angepasst werden. $host wird dabei durch den Namen ersetzt, den der Client aufgerufen hat.

```
access_log /var/log/nginx/$host.access.log;
```

mit „nginx -t“ wird die Konfiguration getestet und mit „nginx -s reload“ geladen.

nginx page caching

Bevor das Caching im Nginx konfiguriert wird, müssen die Seiten, die zwischengespeichert werden sollen oder die Upstream-Webserver, zuerst die passenden Header ausliefern. Das kann z.B. so aussehen:

```
Cache-Control: public, max-age=3600, must-revalidate
```

Was bedeutet das nun?

- public: die Datei ist nicht auf einen bestimmten User zugeschnitten, sondern kann an alle ausgeliefert werden
- max-age: Maximale Lebensdauer in Sekunden
- der Webserver darf den Content nicht mehr ausliefern, bevor er nicht eine auf eine neue Version geprüft hat und ein „304 Not Modified“ erhalten hat

Des weiteren sollte die Applikation nach Ablauf der Lebensdauer des gecachten Objekts korrekt antworten. D.h. es wird der Statuscode „304 Not Modified“ gesendet, wenn sich die Datei nicht geändert hat. Alternativ wird die Seite in der neuen Version geschickt (Code 200 OK) oder „404 File Not Found“ sollte es sie inzwischen nicht mehr geben.

Nginx wird jetzt so konfiguriert, dass er die Dateien in einem Filesystemcache zwischenspeichern kann. Idealerweise legt man den Cache in eine Ramdisk.

```
http {
  # ...
  proxy_cache_path /ramdisk/nginx_cache levels=1:2 keys_zone=cache:5m max_size=100m;

  server {
    # ...

      proxy_pass              http://upstream;
      proxy_cache             cache;
      proxy_cache_key         "$host$request_uri";
      proxy_cache_revalidate  on;
      # Optionally;
      # proxy_cache_use_stale error timeout invalid_header updating http_500 http_502 http_503
http_504;
    }
  }
}
```

Um so z.B. Bilder oder andere statische Dateien durch den Proxy-Nginx cachen zu lassen, kann im Upstream-Nginx diese Einstellung vorgenommen werden:

```
location ~* \.(js|css|png|jpg|jpeg|gif|ico)$ {
  expires max;
  log_not_found off;
}
```

nginx

Permanente Umleitung auf HTTPS

Es gibt mehrere Möglichkeiten auf HTTPS umzuleiten. Hier sind die Gängigsten aufgezeigt.

Per return im http-Serverteil, ist auch die schönste Lösung:

```
server {
  listen 80;
  listen [::]:80;
  server_name seite-x.de;
  return 301 https://$server_name$request_uri;
}
```

Eine unschöne Version mit if (ist dann nützlich wenn man den http- und https-Serverteil kombiniert):

```
if ($scheme = http) {
  return 301 https://$server_name$request_uri;
}
```

Oder per rewrite (auch nicht so schön, da die Domain 2x angegeben wird):

```
server {
  listen 80;
  server_name seite-x.de;
  rewrite ^/(.+) https://seite-x.de/$1 permanent;
}

Wenn alle Webseiten von HTTP auf HTTPS umgeleitet werden sollen ist dieses Beispiel nützlich.
Es funktioniert als Standard-Vhost für alle Seiten, für die es keine explizite Konfiguration
gibt:
<code>
server {
  listen 80 default_server;
  listen [::]:80 default_server;
  server_name _;
  return 301 https://$host$request_uri;
}
```

nginx

Query-String an PHP-FPM mit übergeben

Beim Anflanschen von PHP an nginx habt ihr vermutlich so etwas verwendet:

```
location / {
  try_files $uri $uri/ /index.php;
}
```

Damit auch der Query-String übergeben wird, sollte es folgendermaßen angepasst werden:

```
location / {
  try_files $uri $uri/ /index.php$is_args$query_string;
}
```

nginx

Reverse Proxy

Dieser Artikel beschreibt wie man den nginx-Webserver als Proxy vor einen Apache-Server (oder z.B. auch Tomcat) setzt.

Der Apache muss dafür so konfiguriert werden, dass er z.B. nur auf 127.0.0.1:8080 lauscht und nicht den Port 80 blockiert (/etc/apache2/ports.conf).

Im gewünschten nginx-Vhost werden dann die folgenden Einträge gesetzt:

```
server {
  listen       *:80;
  server_name  meineseite.de;

  location / {
    include proxy_params;
    proxy_pass http://127.0.0.1:8080/;
  }
  ...
}
```

Das include-File /etc/nginx/proxy_params sollte so aussehen (kann dann auch für andere Reverse-Proxies wiederverwendet werden):

```
proxy_redirect off;
proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-Host $host;
#proxy_set_header X-Forwarded-Server $host;
proxy_set_header X-Forwarded-For $remote_addr;
#proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;

proxy_connect_timeout 300;
proxy_send_timeout    300;
proxy_read_timeout    300;
send_timeout          300;
```

Die nginx-Config kann mit „nginx -t“ getestet und mit „nginx -s reload“ neu geladen werden. Ab dann sollten alle Zugriffe auf „meineseite.de“ per Reverse-Proxy von 127.0.0.1:8080 geladen werden.

nginx

Reverse Proxy um HTTP Basic Auth zu entfernen

Manchmal kann es nützlich sein eine mit HTTP Basic Auth geschützte Seite auch ohne Anmeldung (z.B. automatisiert) abzufragen. So blende ich z.B. in meiner Visualisierung der Hausautomation (mit ioBroker) immer ein Livebild einer Webcam ein. Das Bild wird aber durch den Browser geladen (`img src=`). Bis Chrome 59 ging das noch per `http://<user>:<pass>@cam/bild.jpg`. Das wird ab nun, wohl aus Sicherheitsgründen, verhindert. Eine Alternative wäre es die Anmeldedaten für die Cam bei jedem Aufruf wieder einzugeben, ist aber unpraktisch. Daher übernimmt das nun ein kleiner nginx Reverse-Proxy für mich.

Die Lösung ist eigentlich einfach, es muss nur ein zustätzlicher Header an die Cam mitgeschickt werden: `Authorization „Basic dXNlcm5hbWU6cGFzc3dvcnQ=“`

Mein Proxy-Vhost lauscht auf Port 81, die Cam auf Port 80.

Der Base64-codierte Hash wird so erzeugt: `echo -n „username:passwort“ | base64`

Der Hash in diesem Beispiel sieht so aus: `dXNlcm5hbWU6cGFzc3dvcnQ=`

Das -n ist wichtig, sonst wird noch ein Zeilenumbruch angehängt und es funktioniert nicht.

```
#
# reverse proxy with authentication for meine-cam.de
#

server {
  listen 81;
  listen [::]:81;

  root /var/www/html;
  index index.html index.htm index.nginx-debian.html;
  server_name <hostname>;

  location / {
      proxy_pass http://meine-cam.de:80;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header Authorization "Basic <base64 hash einfügen>";
  }
```

```
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
}
```

Da damit natürlich die Authentifizierung umgangen wird, sollte man seinen Reverse Proxy anderweitig einschränken und zum Beispiel den Zugriff nur von einer bestimmten IP/Netz erlauben.

nginx

Rewrite einer Datei auf Datei-$Hostname

Angenommen man will mit einem Vhost mehrere Domains behandeln, da sich z.B. alle einen Documentroot teilen, aber für jeden Auftritt ein eigenes Design per CSS-Datei ausliefern, dann kann man sich mit diesem einfachen Rewrite helfen:

```
rewrite ^/design.css$ /css/$host.css permanent;
```

Jeder Aufruf von z.B. „http://foobar.meine-domain.de/design.css“ wird dann intern umgeleitet auf „http://foobar.meine-domain.de/css/foobar.meine-domain.de.css“.

nginx

Sichere SSL Konfiguration

Mit dem „Mozilla SSL Configuration Generator“ läßt sich aktuell wohl am einfachsten und schnellsten eine sichere Konfiguration für den nginx-Webserver erstellen.

Einfach Webserver, Version von Webserver und OpenSSL setzen und zwischen verschiedenen Cipherprofilen wählen, fertig ist eine Beispielkonfiguration.

Der Eintrag zu TLS im Mozilla Wiki ist auch sehr lesenswert.

> Alternativ habe ich hier noch meine (**veraltete**) Anleitung:

Die folgenden Empfehlungen richten sich nach den Empfehlungen von BetterCrypto.org und ihrem Applied Crypto Hardening PDF mit dem Stand vom 21.04.2016.

Folgendes kann man machen, um die Standard-SSL Konfiguration von nginx sicherer zu gestalten:

- Schwache Cipher und Protokolle abschalten
- Webserver soll die Reihenfolge der Ciphers vorgeben
- SSL-Komprimierung ausschalten
- HSTS aktivieren
- Permanente Umleitung von HTTP nach HTTPS aktivieren

Das folgende gilt für nginx unter Debian. Ich trage dabei nur die beiden Direktiven für SSL-Zertifikatsbundle (ssl_certificate) und SSL-Keyfile (ssl_certificate_key) in den jeweiligen server-Block ein. Die restliche SSL-Konfiguration setze ich global in /etc/nginx/nginx.conf (dort gibts bereits einen Teil „SSL Settings“, das Folgende einfach hinzufügen):

```
ssl_prefer_server_ciphers on;
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_ciphers
'EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+SHA256:EECDH:+CAMELLIA128:+AES128:+SSLv3:!
aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!DSS:!RC4:!SEED:!IDEA:!ECDSA:kEDH:CAMELLIA128-
SHA:AES128-SHA';
ssl_dhparam dhparams.pem;
```

Auf Diffie-Hellman aufsetzende SSH-Cipher benötigen eine entsprechende Parameterdatei (ssl_dhparam). Die Standardgröße beträgt 1024 bit. Die Empfehlung nach der Logjam-Attacke sind mind. 2048 bit, besser 4096 bit. Die Datei wird folgendermaßen erstellt (bei 4096 bit dauerte es ca. 5-15 Minuten):

```
# 4096 bit
openssl dhparam -out dhparams.pem 4096
```

```
# 2048 bit
openssl dhparam -out dhparams.pem 2048
```

Danach muss nginx neu geladen/neu gestartet werden!

Tuning nginx

nginx Tuning

Nach Änderungen nginx neu laden (nginx -t && nginx -s reload):

```
# This number should be, at maximum, the number of CPU cores on your system.
worker_processes 16;

# Number of file descriptors used for Nginx. Should also be set in /etc/security/limits.conf
worker_rlimit_nofile 200000;

# only log critical errors
error_log /var/log/nginx/error.log crit

# Determines how many clients will be served by each worker process.
# (Max clients = worker_connections * worker_processes)
# "Max clients" is also limited by the number of socket connections available on the system (~64
worker_connections 4000;

# essential for linux, optmized to serve many clients with each thread
use epoll;

# Accept as many connections as possible, after nginx gets notification about a new connection.
# May flood worker_connections, if that option is set too low.
multi_accept on;

# Caches information about frequently accessed files. Try to experiment with those values.
open_file_cache max=200000 inactive=20s;
open_file_cache_valid 30s;
open_file_cache_min_uses 2;
open_file_cache_errors on;

# Disable access logs
access_log off;

# Sendfile copies data between one FD and other from within the kernel.
# More efficient than read() + write(), since the requires transferring data to and from the use
sendfile on;

# Tcp_nopush causes nginx to attempt to send its HTTP response head in one packet,
# instead of using partial frames. This is useful for prepending headers before calling sendfile
# or for throughput optimization.
tcp_nopush on;

# don't buffer data-sends (disable Nagle algorithm). Good for sending frequent small bursts of d
tcp_nodelay on;

# Timeout for keep-alive connections. Server will close connections after this time.
keepalive_timeout 30;

# Number of requests a client can make over the keep-alive connection. This is set high for test
keepalive_requests 100000;

# allow the server to close the connection after a client stops responding. Frees up socket-asso
reset_timedout_connection on;

# send the client a "request timed out" if the body is not loaded by this time. Default 60.
client_body_timeout 10;

# If the client stops reading data, free up the stale client connection after this much time. De
send_timeout 2;

# Compression. Reduces the amount of data that needs to be transferred over the network
gzip on;
gzip_min_length 10240;
gzip_proxied expired no-cache no-store private auth;
gzip_types text/plain text/css text/xml text/javascript application/x-javascript application/xml
```

```
gzip_disable "MSIE [1-6]\.";
```

TCP-Stack Tuning

Nach Änderungen „sysctl –system“ ausführen:

```
# Increase system IP port limits to allow for more connections
net.ipv4.ip_local_port_range = 2000 65000

net.ipv4.tcp_window_scaling = 1

# number of packets to keep in backlog before the kernel starts dropping them
net.ipv4.tcp_max_syn_backlog = 3240000

# increase socket listen backlog
net.core.somaxconn = 3240000
net.ipv4.tcp_max_tw_buckets = 1440000

# Increase TCP buffer sizes
net.core.rmem_default = 8388608
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216

net.core.default_qdisc=fq
net.ipv4.tcp_congestion_control=bbr
```

nginx

X-Frame-Options setzen

Über den X-Frame-Options Header (RFC 7034) kann eine Webseite den Browser anweisen, dass sie nicht (ggf. entfernt) über <frame> oder <iframe> geladen werden darf. Das soll sog. Clickjacking verhindern.

In nginx läßt sich das so setzen (innerhalb eines server-Blocks):

```
add_header X-Frame-Options "DENY";
```

Danach nginx reloaden/neu starten.

Folgende Optionen sind möglich:

- DENY ? Es ist keinerlei Einbettung per Frame erlaubt, egal welche Seite es versucht.
- SAMEORIGIN ? Es ist nur die Einbettung in Seiten vom gleichen Ursprung erlaubt.
- ALLOW-FROM ? Es ist nur die Einbettung in Seiten vom angebenen Ursprung erlaubt.

> aktuell wird ALLOW-FROM nur von Firefox und Internet Explorer / Edge unterstützt.

nginx

Zugriff auf .git und .svn Unterverzeichnisse verbieten

Mit diesem Einzeiler (in jeden server{}-Block einfügen) läßt sich der Zugriff auf eventuell vorhandene .git und .svn Verzeichnisse unterbinden:

```
location ~ /\.(svn|git) { deny all; }
```