Sicherheit

- Sicherheit: TLS-Protokoll absichern
- Eigene CA und lokales Zertifikat erstellen

Sicherheit: TLS-Protokoll absichern

In der Basiskonfiguration erlaubt der Server viele mittlerweile als unsicher eingestufte Protokolle und Cipher. Zudem wird nicht zwingend eine Transport-Sicherheit verlangt. Ein Artikel in der c't 2/2022 zu diesem Thema (Security: TLS optimieren) veranschaulicht die Risiken und gibt Lösungsvorschläge an die Hand. besonders erwähnenswert sind dabei die Tools für den Sicherheitschek der Domain (https://ssllabs.com/ssltest) und die Konfigurationsunterstützung (https://ssl-config.mozilla.org).

Im Wesentlichen sind im Falle des V-Servers mit Ubuntu 20.04 und einem Zertifikat von LetsEncrypt für die Domain mykoelle.de zwei Dateien der Apache Konfiguration anzupassen das Apache Modul Headers aktivieren, damit der Server die begehrte A+ Bewertung von im Sicherheitscheck erhält.

Bei der ersten Datei handelt es sich um die vom LetsEncrypt certbot erstellte Datei */etc/letsencrypt/options-ssl-apache.conf*. Durch die Änderung dieser Datei wird sie nicht mehr durch den certbot aktualisiert. Daher muss regelmäßig das Log des certbot (/var/log/letsencrypt/letsencrypt.log) auf updates für diese Datei überprüft werden. Hie die komplette Datei mit Stand 06.01.2022:

```
# This file contains important security parameters. If you modify this file
# manually, Certbot will be unable to automatically provide future security
# updates. Instead, Certbot will print and log an error message with a path to
# the up-to-date file that you will need to refer to when manually updating
# this file.

SSLEngine on

# Intermediate configuration, tweak to your needs

#-------------------------
# Original Cipher configuguration
#
#SSLProtocol             all -SSLv2 -SSLv3
#SSLCipherSuite          ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-
ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-
```

```
AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-
AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-
CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-
SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
#SSLHonorCipherOrder     on
#SSLCompression          off
#
#-------------------------

#-------------------------
# Changed (see ssl-config.mozilla.org and ssllabs.com/ssltest)
#
SSLProtocol             all -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite          ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-
CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384
SSLHonorCipherOrder     off
SSLSessionTickets       off
Protocols               h2 http/1.1
Header                  always set Strict-Transport-Security "max-age=63072000"
#
#-------------------------

SSLOptions +StrictRequire

# Add vhost name to log entries:
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" vhost_combined
LogFormat "%v %h %l %u %t \"%r\" %>s %b" vhost_common

#CustomLog /var/log/apache2/access.log vhost_combined
#LogLevel warn
#ErrorLog /var/log/apache2/error.log
```

Damit im Header eine zwingende Transport-Sicherheit eingetragen werden kann, muss das Modul Headers des Apache Webservers aktiviert werden. Dies geschieht mit dem Befehl `sudo a2enmod headers`.

Für eine Optimierung sorgen die beiden folgenden Zeilen in der Datei */etc/apache2/sites-enabled/bookstack-le-ssl.conf*. Sie sind unterhalb des </VirtualHost> Tags, also außerhalb der

VirtuaHost Beschreibung einzufügen:

```
<VirtualHost *:443>
....
</VirtualHost>
#
# Added (see ct 2/2022 Security: TLS optimieren und ssl-config.mozilla.org and
ssllabs.com/ssltest)
#
SSLUseStapling          On
SSLStaplingCache        "shmcb:logs/ssl_stapling(32768)"
```

Nun kann mit den Anweisungen `sudo apachectl configtest` und `sudo apachectl restart` der Apache Webserver neu gestartet werden. Ein Test der Domain via https://ssllabs.com/ssltest sollten nun ein grünes A+ ergeben.T

Eigene CA und lokales Zertifikat erstellen

Für Tests ist es sinnvoll, Serveranwendungen auch unter https zu testen. Um das auf dem Entwicklungsrechner zu tun, benötigt man ein lokales Zertifikat und möglichst auch eine eigene CA, damit das Zertifikat auch ohne die lästigen Warnungen akzeptiert wird.

Auf einem Mac oder jedem anderen System mit Homebrew kann das mit brew erledigt werden. Zuerst aktualisiert man brew:

```
brew update && brew upgrade
```

Danach installiert man das Tool mkcert:

```
brew install mkcert
```

Damit mkcert das neu erstellte Zertifikat bzw. die CA in Firefox installieren kann, benötigt mkcert noch das Tool certutil aus dem Cask nss. Wird also auch mit Firefox getestet, ist nss noch zu installieren:

```
brew install nss
```

Nun kann eine CA und ein Zertifikat für localhost erstellt werden:

```
mkcert -install
mkcert localhost 127.0.0.1 ::1
```

Der Befehl `mkcert -install` kann übrigens beliebig oft wiederholt werden.

Um die erzeugten <name>.pem- und <name-key>.pem Dateien in <name>.crt und <name>.key Dateien umzuwandeln, wird openssl genutzt.

```
openssl x509 -in localhost.pem -out localhost.crt
openssl pkey -in localhost-key.pem -out localhost.key
```

Einsatz z. B. in einer Go-Anwendung mit dem Echo Framework:

```
func main() {
e := echo.New()
```

```
	// add middleware and routes
	// ...

	sc := echo.StartConfig{Address: ":1323"}
	if err := sc.StartTLS(context.Background(), e, "localhost.crt", "localhost.key"); err != nil {
		e.Logger.Error("failed to start server", "error", err)
	}
}
```