Labreport #6

Patrick Eickhoff, Alexander Timmermann

Aufgabe 1: Absicherung des TCP-Chats mit SSL

Aufgabe 2: CAs und Webserver-Zertifikate

2. Um ein selbstsigniertes Zertifikat zu erzeugen, sind im wesentlichen drei OpenSSL-Befehle notwendig:

```
$ openssl genrsa -out server.key 2048
$ openssl req -new -key server.key -out server.csr
$ openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Mit dem ersten Befehl erzeugen wir einen neuen, 2048-bit langen privaten Schlüssel. Wir erstellen dann mit dem zweiten Befehl die sog. *certificate signing request*, in der die Informationen enthalten sind, die das Zertifikat später umfassen soll. Diese werden interaktiv abgefragt. Als Common Name verwenden wir `vmsrv11.svslab`.

Im dritten Befehl verwenden wir schließlich den erstellten Private Key, um aus der CSR ein Zertifikat zu generieren, das 365 Tage gültig ist.

Um dieses Zertifikat für den Apache-Server einzusetzen, ersetzen wir in der Konfigurationsdatei die Pfade hinter `SSLCertificateFile` und `SSLCertificateKeyFile` mit den Pfaden zum Zertifikat und dem private key.

Dass selbst-signierte Zertifikate als unsicher angesehen werden liegt auf der Hand. Jeder kann sich ein Zertifikat erstellen, was bspw. für google.com gültig ist. Würde diesen Zertifikaten vertraut, ließe sich eine Man-in-the-Middle-Attacke zu google.com sehr leicht implementieren, da der User keinen Unterschied bemerkt, und ihm durch das Zertifikat weitere Sicherheit suggeriert wird.

Die Aufgabe einer CA ist es, diese Angaben zu validieren und im Zweifelsfall die Ausstellung eines Zertifikats zu verweigern. CAs sind deshalb Firmen oder Organisationen, denen vertraut wird, diese Macht nicht zu missbrauchen.

3. Um mit Hilfe von *mod_rewrite* eine Weiterleitung zu HTTPS zu realisieren, fügen wir folgende Optionen in die Apache-Config ein:[1]

```
1 RewriteEngine On
2 RewriteCond %{HTTPS} !=on
3 RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]
```

Mit dem ersten Befehl wird die Bearbeitung der Rewrite-Regeln eingeschaltet. Danach wird geprüft, ob nicht bereits HTTPS verwendet wird, da damit die Weiterleitung hinfällig wird. Schließlich wird die Rewrite-Regel definiert, die dann weiterleitet. Die Optionen am Ende sind die Flags. `L` gibt an, dass die Regel die letzte ist, die ausgeführt wird. `R` spezifiziert, dass ein HTTP Redirect passieren soll.

[1] Quelle: `https://wiki.apache.org/httpd/RewriteHTTPToHTTPS`

Die Regel selbst ist ein regulärer Ausdruck, bei dem alles nach einem optionalen "/" matched und in die HTTPS-URL eingesetzt wird.

4. Über die verschlüsselte Verbindung können wir die Zugangsdaten selbstverständlich nicht mitlesen. Verwenden wir *sslstrip* als Proxy, so wird die Umleitung auf die verschlüsselte Verbindung verhindert und wir können die Zugangsdaten recht komfortabel im Log-File nachlesen.

   Die Sicherheit der Weiterleitung ist damit maximal als "gut gemeinte" Maßnahme für Endanwender gedacht, taugt aber nicht, ernsthafte Angriffe abzuwehren. Hierzu sind weitere Maßnahmen nötig, wie beispielsweise HSTS.

   HTTP Strict Transport Security (HSTS) ist ein in RFC 6797[2] beschriebener Standard, der einen solchen Angriff verhindern kann. Wird HSTS aktiviert, so können Seiten unter einer Domain, die HSTS setzt, nicht über eine unverschlüsselte Verbindung abgerufen weden. HSTS wird über einen Header aktiviert, der bei der Antwort mitgesendet wird und etwa so aussieht:

```
1 Strict-Transport-Security: max-age=15768000
```

   Damit wird HSTS im Browser strikt forciert und lässt sich auch für den Endnutzer nicht wieder abschalten. Mit dem `max-age`-Parameter wird eine Art "Haltbarkeitsdatum" in Sekunden angegeben. 15768000 Sekunden entsprechen etwa einem halben Jahr, danach wird der Browser eine unverschlüsselte Verbindung prinzipiell wieder erlauben, wenn nicht erneut ein HSTS-Header mitgesendet wird.

Aufgabe 3: Unsichere selbstentwickelte Verschlüsselungsalgorithmen

[2] `https://tools.ietf.org/html/rfc6797`