Security Lab – Zertifikate und TLS

VMware

Dieses Lab müssen Sie mit dem **Ubuntu-Image** durchführen.

1 Einleitung

In diesem Praktikum werden Sie sich mit X.509-Zertifikaten und dem TLS-Protokoll genauer auseinandersetzen. Der Fokus liegt dabei bei den Zertifikaten, denn diese sind für die Kommunikationssicherheit im Internet von grösster Bedeutung und werden sehr oft eingesetzt. Deshalb sollten alle Informatikstudierenden das Prinzip und die Anwendung der Zertifikate verstehen.

Entsprechend ist es wichtig, dass Sie die einzelnen Aufgaben sorgfältig bearbeiten, sich jeweils genau überlegen, was Sie gerade tun und sich auch Ihre Antworten für die diversen Fragen gut überlegen. Nur so werden Sie am Ende des Praktikums das recht komplexe Gebiet der digitalen Zertifikate «gemeistert» haben.

Das Praktikum besteht aus drei Teilen:

- Im ersten und umfassendsten Teil (Abschnitt 3) werden Sie eigene Schlüsselpaare und Zertifikate erstellen und diese sowohl im Apache-Webserver als auch im Browser installieren und konfigurieren, damit gesicherte Verbindungen mit TLS (bzw. HTTPS) aufgebaut und verwendet werden können. Dabei werden Sie einiges über den Inhalt von Zertifikaten, die Prüfung von Zertifikaten und den praktischen Einsatz von Zertifikaten lernen.
- Im zweiten Teil (Abschnitt 4) werden Sie den TLS-Verbindungsaufbau genauer analysieren, um nachzuvollziehen, was dabei im Detail passiert ist. Dies wird Ihnen helfen, das TLS-Protokoll und die Verwendung der Zertifikate im Rahmen dieses Protokolls besser zu verstehen.
- Im dritten Teil (Abschnitt 5) wird auf einige weitere Aspekte von Zertifikaten eingegangen, die in den ersten beiden Teilen noch nicht adressiert wurden.

Als Basis werden in Abschnitt 2 zuerst einige Grundlagen im Zusammenhang mit Zertifikaten eingeführt. Diesen Teil sollten Sie aus der Vorlesung bereits kennen und wenn Sie dort grössere Verständnisschwierigkeiten haben, dann sollten Sie zuerst die Vorlesungsunterlagen nochmals genau studieren.

Führen Sie sämtliche Konfigurationen (ausser den Konfigurationen in Firefox) und Befehle in diesem Praktikum als Benutzer *root* durch. Verwenden Sie *su -* in einem Terminal, um *root* zu werden.

2 Wichtigste Grundlagen zu Zertifikaten

Als Basis für dieses Praktikum sind nochmals die wichtigsten Grundlagen im Zusammenhang mit Zertifikaten zusammengefasst:

- Ein Zertifikat dient primär dazu, einen Public Key an eine Identität - z.B. einen Hostnamen wie www.zhaw.ch - zu binden. Dies ist eine entscheidende Grundtechnologie, um beliebige Kommunikationspartner über ein Netzwerk zu authentisieren (z.B. als Teil von TLS).
- Zertifikate werden von Certification-Authorities (CA) ausgestellt. Es gibt öffentliche CAs, man kann aber auch eine eigene (z.B. firmenintern oder für private Zwecke) verwenden. Damit nicht einfach jeder ein Zertifikat im Namen einer CA ausstellen kann, werden ausgestellte Zertifikate von der CA signiert. Dazu besitzt die CA ein Schlüsselpaar bestehend aus Public Key und Private Key. Für die Signierung von Zertifikaten wird der Private Key verwendet.
- CAs haben selbst auch Zertifikate, die die Identität der CA und den Public Key der CA beinhalten. Diese Zertifikate sind self-signed, d.h. sie sind mit dem Private Key der CA signiert.
- Damit eine Softwarekomponente (z.B. ein Browser) bei einem erhaltenen Zertifikat prüfen kann, ob dieses wirklich von einer CA ausgestellt (und damit signiert) wurde, wird der Public Key der CA benötigt – denn nur mit dem passenden Public Key kann die Signatur verifiziert und damit die

Echtheit eines Zertifikats bestätigt werden. Dazu muss die Softwarekomponente den Public Key bzw. das Zertifikat der CA kennen. Dies löst man in der Praxis so, dass diese Zertifikate in den Softwarekomponenten (in der Applikation selbst oder im unterliegenden Betriebssystem) vorinstalliert werden.

Nachfolgend ist das Grundprinzip beschrieben, wie ein Client einen Server mit Hilfe von Zertifikaten authentisieren kann. In den eigentlichen Protokollen (z.B. TLS) sieht das im Detail dann jeweils etwas anders aus, das Grundprinzip ist aber immer dasselbe (siehe auch die Vorlesungsunterlagen):

- Der Client sendet dem Server eine Challenge.
- Der Server signiert die Challenge mit seinem Private Key, was die Response ergibt.
- Der Server sendet die Response und sein Zertifikat zum Client.
- Der Client prüft, ob das Zertifikat des Servers von einer vertrauenswürdigen CA ausgestellt wurde (unter Verwendung des Public Key des passenden vorinstallierten CA-Zertifikats).
- Ist das Zertifikat gültig, so weiss der Client, dass der Public Key im Zertifikat zur Identität im Zertifikat gehört.
- Der Client prüft, ob die Identität im Zertifikat (z.B. ein Hostname) der Identität entspricht, welche er für die Verbindungsaufnahme mit dem Server verwendet hat. Falls dem so ist, so gehört der Public Key im Zertifikat dem Server, mit dem der Client sprechen möchte. Der Client weiss nun also, dass er den richtigen Public Key hat.
- Nun wird die Response (im Wesentlichen eine Signatur) überprüft, mit Verwendung des Public Keys aus dem Zertifikat.
- Kann dies erfolgreich gemacht werden, d.h. entspricht die Response wirklich einer Signatur über der Challenge, dann beweist dies dem Client, dass der Server den Private Key, der zum Public Key im Zertifikat gehört, kennt – sonst hätte er die Response nicht korrekt erzeugen können.
- Dies beweist dem Client, dass der Server der richtige Server ist, denn nur dieser kennt diesen Private Key (unter Annahme, dieser wurde nie gestohlen oder geknackt).

3 Eigene Schlüsselpaare und Zertifikate erstellen und verwenden

In diesem Teil werden Sie mit *openssl* eigene Schlüsselpaare und Zertifikate erstellen und diese sowohl im Apache-Webserver als auch im Browser installieren und konfigurieren, um anschliessend gesicherte Verbindungen mit TLS (bzw. HTTPS) aufbauen und verwenden zu können. Im Detail werden Sie zuerst ein Schlüsselpaar und ein zugehöriges self-signed Zertifikat für eine eigene CA erzeugen. Danach erzeugen Sie ein Schlüsselpaar für den Webserver und stellen mit Ihrer CA ein Zertifikat für den Webserver aus. Schliesslich werden Sie auch noch den Client (Browser) mit einem Schlüsselpaar und zugehörigem Zertifikat ausrüsten, damit eine beidseitig authentisierte TLS-Kommunikationsbeziehung möglich wird.

openssl verwendet ein Konfigurationsfile */etc/ssl/openssl.cnf*, das Voreinstellungen für die zu generierenden Zertifikate enthält. Es spezifiziert zum Beispiel, welche Extensions (mehr dazu weiter unten) Zertifikate haben sollen und welche Werte diese erhalten. Setzen Sie das für dieses Praktikum zu verwendende File, indem Sie in einem Terminal (als *root*) folgendes ausführen:

```
$ /securitylab/certificates/setSL.sh
```

Die Befehle zum Erzeugen der Zertifikate werden Ihnen vorgegeben. Falls Sie jedoch weitere Informationen haben möchten, so erhalten Sie dies über die Manpages von *openssl* oder *http://www.openssl.org*.

3.1 Schlüsselpaar und self-signed Zertifikat für die CA erzeugen

Im ersten Schritt erzeugen Sie ein Schlüsselpaar für die CA und ein zugehöriges self-signed Zertifikat. Wenn Sie ein Zertifikat von einer offiziellen CA beziehen, dann würden Sie diesen Schritt natürlich nicht ausführen. Wenn Sie aber firmenintern oder für private Zwecke eine CA betreiben möchten, dann würden Sie im Wesentlichen so vorgehen wie nachfolgend beschrieben.

Das Erzeugen eines Schlüsselpaars und zugehörigem selbstsigniertem Zertifikat kann mit einem einzigen Befehl durchgeführt werden. Führen Sie diesen – wie alle weiteren *openssl* Befehle auch, in einem Terminal im Verzeichnis */etc/ssl* aus (als *root*):

```
$ openssl req -x509 -days 1460 -newkey rsa:2048 -keyout caKey.pem
  -out caCert.pem
```

Die *openssl* Parameter spezifizieren dabei die Gültigkeitsdauer (1'460 Tage = 4 Jahre) des Zertifikats, die Art des Schlüsselpaars (RSA) und die Länge der erstellten Schlüssel (2'048 Bits). Der Private Key wird in *caKey.pem* und das Zertifikat (inkl. dem Public Key) in *caCert.pem* abgespeichert. Geben Sie Acht, dass Sie genau diese Filenamen verwenden, damit alles mit den Einstellungen in *openssl.cnf* korrekt zusammenspielt.

Nachdem Sie den Befehl ausgeführt haben, müssen Sie diverse Informationen angeben:

- Eine Passphrase[1], mit der der Private Key geschützt (verschlüsselt) wird. Wählen Sie hier die Passphrase *ca-passphrase*, um diese eindeutig von anderen Passphrases, die noch folgen werden, zu unterscheiden. Es versteht sich von selbst, dass in der Realität eine viel stärkere Passphrase gewählt werden sollte.
- Informationen zur Identität (X.500 Name) der CA. Geben Sie hier bei *Country Name* den Wert *CH* ein, bei *Organization Name* den Wert *ZHAW* und bei *Common Name* den Wert *GruppeX_CA*, wobei *X* Ihrer Gruppennummer entspricht. *E-Mail Address* können Sie leer lassen.

Verschieben (*mv*-Befehl) Sie den erzeugten Private Key anschliessend nach */etc/ssl/private* und das Zertifikat nach */etc/ssl/cacerts*., auch dies ist notwendig, damit alles korrekt mit den Einstellungen in *openssl.cnf* zusammenspielt.

Schauen Sie nun den rohen Inhalt des Zertifikats im Terminal an. Verwenden Sie dazu den folgenden Befehl:

```
$ less cacerts/caCert.pem
```

Mit den dargestellten Daten können Sie nicht viel anfangen, denn Zertifikate werden nicht in einer für den Menschen lesbaren Form gespeichert, sondern sind sogenannt PEM-codiert. Daneben gibt es auch andere Codierungsvarianten wie z.B. DER-Codierungen.

openssl bietet die Möglichkeit, den Inhalt von Zertifikaten in lesbarer Form darzustellen. Verwenden Sie dazu den folgenden Befehl:

```
$ openssl x509 -in cacerts/caCert.pem -noout -text
```

Schauen Sie sich die Information im Zertifikat zuerst einmal komplett von oben bis nach unten an, um einen Überblick des Inhalts zu erhalten. Dabei erkennen Sie unter anderem, dass für *Issuer* und *Subject* die von Ihnen eingegebenen Werte verwendet werden und dass die Gültigkeitsdauer den spezifizierten 4 Jahren «ab jetzt» entspricht.

Sie sehen zudem, dass das Zertifikat neben den Grunddaten, die in jedem Zertifikat vorhanden sein müssen, auch sogenannte *Extensions* verwendet. Ganz wichtig dabei ist *Basic Constraints*: Dieses Feld bezeichnet, ob ein Zertifikat ein CA-Zertifikat ist oder nicht. Bei diesem Zertifikat ist der Wert

[1] *Passphrase* wird oft verwendet, um ein besonders langes Passwort (z.B. ein ganzer Satz) zu bezeichnen.

CA:TRUE, was bedeutet, dass dies ein CA-Zertifikat ist und deshalb verwendet werden kann, um weitere Zertifikate auszustellen (zu signieren) und zu verifizieren.

Beantworten Sie folgende Fragen zum dargestellten Inhalt des Zertifikats. Dazu müssen Sie ggfs. die Vorlesungsunterlagen oder auch das RFC[2] zu X.509 Zertifikaten konsultieren:

Das Zertifikat beinhaltet zwei Identitäten, den *Issuer* und das *Subject*. Was bedeuten *Issuer* und *Subject* in einem Zertifikat ganz grundsätzlich bzw. was wird damit bezeichnet?

Issuer: Die Person oder das Unternehmen, die/das das Zertifikat ausgestellt hat bzw. signiert hat.

Subject: Die Person oder das Unternehmen, welche/s das Zertifikat besitzt.

Issuer und *Subject* weisen in diesem Zertifikat den gleichen Wert auf. Wieso ist das so?

Da wir gleichzeitig das Zertifikat ausgestellt haben und uns die Domain gehört, ist das Zertifikat self-signed.

Das Zertifikat beinhaltet zwei lange Hex-Strings (jeweils mehrere Zeilen). Was beinhalten diese Strings?

Public-Key vom Subject und die Signatur des Herausgebers

Im Feld *Signature Algorithm* stehen die Namen von zwei verschiedenen kryptographischen Verfahren. Wozu dient dieses Feld und wieso stehen da *zwei* Verfahren drin?

sha256WithRSAEncryption

RSAEncryption ist das Verfahren, wie die Schlüssel verschlüsselt werden.
Dazu wird der encrypted value noch mit SHA256 gehashed.

Wie beurteilen Sie die generelle kryptographische Sicherheit dieser beiden Verfahren (auch unter Berücksichtigung der Schlüssellänge)? Ist es in Ordnung, diese Verfahren heute noch zu verwenden?

[2] https://tools.ietf.org/html/rfc6818

RSA mit 2048 Bits hat schon einen hohen work-factor(2^(2048-1)) und dazu wird dies noch mit SHA mit 256 Bits gehashed und somit doppelt gesichert.

Deshalb ist das Verfahren, auch für den heutigen Standard, ziemlich gut.

3.2 Schlüsselpaar und Zertifikats-Request für den Server erzeugen

Erzeugen Sie nun ein Schlüsselpaar für den Webserver und einen zugehörigen Zertifikats-Request (nach wie vor in einem Terminal im Verzeichnis */etc/ssl* als *root*):

```
$ openssl req -newkey rsa:2048 -keyout serverKey.pem
  -out serverReq.pem
```

Der Befehl ist sehr ähnlich wie der obige, um das Zertifikat für die CA zu erzeugen, es fehlen einzig der Parameter *-x509* und die Gültigkeitsdauer. Als Schlüssel wird wiederum ein RSA-Schlüssel mit einer Länge von 2'048 Bits verwendet. Der Private Key wird in *serverKey.pem* und der Zertifikats-Request in *serverReq.pem* abgespeichert.

Nachdem Sie den Befehl ausgeführt haben, müssen Sie diverse Informationen angeben:

- Eine Passphrase, mit der der Private Key geschützt (verschlüsselt) wird. Wählen Sie hier die Passphrase *server-passphrase*, um diese eindeutig von anderen Passphrases zu unterscheiden.
- Informationen zur Identität (X.500 Name) des Webservers. Geben Sie hier bei *Country Name* den Wert *CH* ein, bei *Organization Name* den Wert *ZHAW* und bei *Common Name* den Wert *host.securitylab.local* – diesen Wert werden wir später als Hostnamen für den Webserver verwenden. *E-Mail Address* können Sie leer lassen.

Wie erwähnt wird hier neben dem Schlüsselpaar ein sogenannter Zertifikats-Request erzeugt. Schauen Sie sich dessen Inhalt mit folgendem Befehl an:

```
$ openssl req -in serverReq.pem -noout -text
```

Auf den ersten Blick sieht das ähnlich aus wie ein Zertifikat, es gibt aber einige Unterschiede. So fehlen z.B. *Issuer*, die *Gültigkeitsdauer* und auch die *Extensions*. Im Wesentlichen sind nur die Informationen erhalten, die der Benutzer spezifiziert hat: Das *Subject* und der *Public Key*. Zudem enthält der Zertifikats-Request eine Signatur über diese Daten, die mit dem Private Key des Webservers erzeugt wurde.

Wozu braucht es diesen Zertifikats-Request überhaupt? Diesen braucht es, um bei einer CA ein Zertifikat zu beantragen. Die CA verwendet dann die Daten im Zertifikats-Request, um daraus das richtige, von der CA signierte Zertifikat auszustellen. In einem realen Szenario würde dieser Zertifikats-Request nun also zu einer CA gesendet, z.B. über das Web-Interface der CA.

Befinden sich in diesem Zertifikats-Request irgendwelche kritischen Daten? D.h. ist es von zentraler Bedeutung, dass dieser Request nur über einen sicheren Kommunikationskanal zur CA gesendet wird? Können Sie sich ein Angriffsszenario vorstellen, wenn der Request ungesichert (d.h. weder verschlüsselt noch authentisiert / integritätsgeschützt) übertragen wird?

Muss nicht unbedingt über einen sicheren Kanal übertragen werden, da die Signatur vom CA überprüft werden kann. Die meisten Angaben sind sowieso dazu ausgelegt, dass das public sichtbar ist.

Betrachten wir die Signatur im Zertifikats-Request etwas genauer: Diese ist mit dem Private Key des Webservers erzeugt worden. Der Zertifikats-Request ist damit auch self-signed, denn er beinhaltet den zugehörigen Public Key.

Wieso ist diese Signatur wichtig? Wieso genügt es nicht einfach, dass man den Public Key und das Subject zur CA sendet, um ein Zertifikat zu erhalten? Erklären Sie das am besten anhand eines Szenarios, das möglich wäre, wenn diese Signatur nicht verwendet würde.

Wenn nur das Public Key gesendet wird, kann dies von einem Attacker ausgetauscht werden. Mit der Signatur hingegen kann man die Public Key überprüfen.

3.3 Zertifikat für den Webserver ausstellen

Der Zertifikats-Request wird nun also der CA übergeben, damit diese ein gültiges Zertifikat generiert. In unserem Fall mit Webserver und CA auf dem gleichen Host entfällt natürlich dieser Schritt der Übergabe des Requests.

Sie nehmen nun also wieder die Rolle der CA ein und erzeugen aus dem Zertifikats-Request das Zertifikat für den Webserver. Dies machen Sie mit folgendem Befehl, der das Zertifikat mit einer Gültigkeitsdauer von einem Jahr ausstellt.

```
$ openssl ca -in serverReq.pem -days 365 -out serverCert.pem
```

Nach Eingabe des Befehls müssen Sie die Passphrase (*ca-passphrase*) eingeben, mit welchem der Private Key der CA geschützt ist, denn dieser wird ja benötigt, um das Zertifikat zu signieren. Anschliessend werden die wichtigsten Informationen des auszustellenden Zertifikats zur Überprüfung angezeigt und Sie können durch zweimalige Eingabe von *y* die Ausstellung komplettieren.

Beachten Sie, dass bei obigem Befehl der Private Key der CA nicht spezifiziert wurde, obwohl dieser natürlich für die Ausstellung des Zertifikats benötigt wird. Der Grund, wieso der Private Key nicht angegeben werden muss, liegt darin, dass dies bereits in *openssl.cnf* konfiguriert ist – deshalb war es wichtig, im vorhergehenden Schritt genau die richtigen Filenamen zu verwenden.

Betrachten Sie nun den Inhalt des erzeugten Zertifikats mit folgendem Befehl:

```
$ openssl x509 -in serverCert.pem -noout -text
```

Sie sehen, dass der Aufbau des Zertifikats weitgehend identisch ist wie der Aufbau des self-signed CA-Zertifikats, das Sie vorhergehend analysiert haben. Als einzigen Unterschied enthält das Zertifikat des Webservers eine zusätzlich Extension *Subject Alternative Name*, die den bei *Common Name* eingegebenen Hostnamen *host.securitylab.ch* enthält. Mehr zu dieser Extension folgt weiter unten. Vergleichen Sie nun den Inhalt des Zertifikats des Webservers mit dem Inhalt des Zertifikats der CA und beantworten Sie folgende Fragen.

Betrachten Sie die Felder *Issuer* und *Subject* in beiden Zertifikaten. Inwiefern besteht hier eine Verbindung über die beiden Zertifikate hinweg?

Das Zertifikat vom Webserver wurde vom CA signiert bzw. beim CA selber wurde es selbst signiert. Deshalb ist die Verbindung durch den Issuer.

Wie kann damit generell festgestellt werden, ob ein Zertifikat potentiell von einem bestimmten CA-Zertifikat ausgestellt wurde? («Potentiell» deshalb, weil natürlich auch noch die Signatur auf dem ausgestellten Zertifikat korrekt sein muss.)

Verstehe die Frage nicht ganz...

Bis man vlt. zum self-signed Zertifikat kommt?

Nebenbei: Wenn wir hier schreiben «ein Zertifikat wird von einem CA-Zertifikat» ausgestellt, dann ist das natürlich nicht ganz korrekt, es wird aber im Sprachgebrauch sehr häufig so verwendet. Effektiv wird für das Ausstellen eines Zertifikats natürlich nicht das CA-Zertifikat benötigt, sondern der Private Key der CA, der das Gegenstück des Public Key im CA-Zertifikat darstellt. Das CA-Zertifikat selbst wird erst bei der Prüfung des ausgestellten Zertifikats benötigt.

Im Zertifikat des Webservers steht bei *Basic Constraints* der Wert *CA:FALSE*. Offensichtlich ist dies also kein CA-Zertifikat. Das korrekte Setzen dieses Werts ist von höchster Wichtigkeit beim Ausstellen eines Zertifikats für einen Server. Wieso ist das so? Welches Angriffsszenario wäre möglich, wenn dieses Feld falsch (also auf *CA:TRUE*) gesetzt würde?

Dann könnte der Webserver auch Zertifikate erstellen, welche dann als vertrauenswürdig von Clients eingestuft werden.

Weiter oben wurde gefragt, ob der Zertifikats-Request irgendwelche sicherheitskritischen Informationen beinhaltet. Wie sieht es nun mit dem ausgestellten Zertifikat aus? Befinden sich in diesem irgendwelche kritischen Daten? Oder anders gefragt, wäre es grundsätzlich in Ordnung, dass die CA das ausgestellte Zertifikat in einer ungesicherten E-Mail-Nachricht an den Antragssteller sendet?

Das Zertifikat wurde genau für diesen Zweck ausgelegt, damit alle Infos public sein können. Deshalb nicht sicherheitskritisch...

Betrachten Sie nochmals den ganzen Prozess, um das Webserver-Zertifikat zu erhalten (Schlüsselpaar generieren, Zertifikats-Request generieren, Zertifikat ausstellen). Und nehmen Sie an, wir verwenden keine eigene CA, sondern eine offizielle (öffentliche) CA, die grundsätzliche Vorgehensweise sei aber gleich wie oben. Was geschieht während dieses ganzen Prozesses mit dem Private Key des Webservers? Verlässt dieser den Rechner des Antragsstellers irgendwann bzw. wird dieser an die CA gesendet?

Der Private Key wird immer geheim gehalten, deshalb bleibt das auf dem Rechner und wird nicht etwa weitergeschickt.

Zusammengefasst wird mit der CA im ganzen Prozess nur öffentliche, nicht-sicherheitskritische Information ausgetauscht und der besonders sicherheitskritische Private Key des Antragsstellers bleibt immer auf seinem eigenen Rechner.

3.4 Zertifikat und Private Key im Webserver installieren und Server starten

Mit den obigen Schritten haben Sie alle notwendigen Schlüssel und Zertifikate erzeugt. Nun werden Sie diese im Apache-Webserver und im Browser installieren und konfigurieren, damit gesicherte Verbindungen mit TLS (bzw. HTTPS) aufgebaut und verwendet werden können.

Das virtuelle Ubuntu-Linux System hat eine private IP-Adresse. Damit Sie lokal dennoch mit einem Hostnamen arbeiten können, öffnen Sie */etc/hosts* mit einem Editor (z.B. *vim* oder *gedit*) und setzten Sie in der Zeile

```
192.168.57.131  host.securitylab.local
```

Ihre aktuelle IP-Adresse ein (dies erhalten Sie mit *ifconfig*). Damit können Sie Ihren Host mit dem Namen *host.securitylab.local* ansprechen.

Um den Apache-Webserver zu konfigurieren, gehen Sie wie folgt vor:

- */etc/apache2/sites-available* beinhaltet die verschiedenen Konfigurationen der von dieser Apache Installation «gehosteten» Virtual Hosts. Ein «physischer» Apache-Webserver kann mehrere Virtual Hosts (logische Webserver) unter verschiedenen Hostnamen oder Ports anbieten. Für dieses Praktikum ist das File *lab_certificates.conf* relevant. Alle anderen Konfigurationsfiles können Sie ignorieren.
- Öffnen Sie das File */etc/apache2/sites-available/lab_certificates.conf* mit einem Texteditor. Passen Sie den *VirtualHost* Eintrag ganz am Anfang an und verwenden Sie den Hostnamen Ihres Rechners. Die Zeile sollte also wie folgt aussehen:

  ```
  <VirtualHost host.securitylab.local:443>
  ```

- Wenn Sie weiter runterscrollen, so finden Sie zwei Einträge *SSLCertificateFile* und *SSLCertificateKeyFile*. Damit sind die Filenamen spezifiziert, die das Zertifikat und den Private Key des Webservers beinhalten. Verschieben Sie die im vorhergehenden Abschnitt im Verzeichnis */etc/ssl* erzeugten Files *serverCert.pem* und *serverKey.pem* an die hier angegebenen Locations.
- Nur zur Information: Die Konfiguration *DocumentRoot* ganz zu Beginn bezeichnet das Root-Verzeichnis für die Webserver-Files – im vorliegenden Fall */var/www/lab_certificates*. Dieses Verzeichnis beinhaltet nur ein einfaches File *index.html* und sonst nichts. Bei einem realen Webserver wären dort natürlich diverse Dokumente wie HTML-Files, Bilder, Scripts etc. abgelegt. An der TLS-Konfiguration ändert das aber nichts, da mit der hier verwendeten Konfiguration sämtli-

che Files und Verzeichnisse unterhalb */var/www/lab_certificates* nun nur über TLS (bzw. HTTPS) erreicht werden können.

Die verschiedenen Virtual Hosts in */etc/apache2/available-sites* können jeder für sich aktiviert und deaktiviert werden. *lab_certificates* ist derzeit noch deaktiviert. Die Aktivierung ist sehr einfach: Es wird ein symbolischer Link in */etc/apache2/enabled-sites* auf die Konfigurationsdatei erzeugt und damit ist der Virtual Host aktiviert. Der Link muss nicht manuell erzeugt bzw. entfernt werden, es existieren dazu zwei Befehle (die Endung *.conf* des Konfigurationsfiles müssen Sie jeweils weglassen):

```
$ a2ensite lab_certificates    # aktiviert den Virtual Host
$ a2dissite lab_certificates   # deaktiviert den Virtual Host
```

Schauen Sie unter */etc/apache2/sites-enabled* nach, ob sich bereits aktivierte Virtual Hosts (symbolische Links) darin befinden. Falls ja, so deaktivieren Sie diese mittels entsprechenden Aufrufen von *a2dissite*.

Aktivieren Sie jetzt den Virtual Host *lab_certificates* mittels oben angegebenem *a2ensite* Befehl. Nach einer Aktivierung/Deaktivierung und auch nach jeder Änderung in den Konfigurationsfiles müssen Sie den Webserver neu starten, damit die Änderungen berücksichtigt werden. Am einfachsten starten/stoppen Sie Apache mit dem folgenden Befehl:

```
$ systemctl (start|stop|restart) apache2
```

Starten Sie Apache also neu (er ist nach einem Neustart des Rechners jeweils bereits gestartet) mit folgendem Befehl:

```
$ systemctl restart apache2
```

Beachten Sie allfälligen Output des Befehls auf der Kommandozeile, da dies auf mögliche Probleme hinweisen kann. Eine allfällige Meldung *Could not reliably determine the server's fully qualified domain name...* können Sie ignorieren. Generell liefert auch das Apache Error-Logfile gute Informationen bei Problemen (*/var/log/apache2/error.log*).

Anschliessend müssen Sie die Passphrase (*server-passphrase*), die den Private Key des Webservers schützt, eingeben, denn der Server muss während des TLS-Verbindungsaufbaus darauf zugreifen können (falls diese Abfrage nicht stattfindet, starten Sie den Server erneut neu). Falls Sie immer noch nicht nach einer Passphrase gefragt werden, so müssen Sie das *mod_ssl*-Modul aktivieren (als *root* im Terminal *a2enmod ssl* ausführen). Greifen Sie nun mit dem Browser auf den Webserver zu:

```
https://host.securitylab.local
```

Es erscheint ein Hinweis «dass die Verbindung nicht sicher ist» Was ist hier passiert? Ein Klick auf den *Advanced*-Button liefert weitere Informationen.

Eine Sicherheitswarnung, dass der Browser die Seite aufgrund des Zertifikates als unsicher empfindet. Man hat die Möglichkeit auf eigenes Risiko trotzdem die Seite zu besuchen.

Welches Sicherheitsrisiko besteht, wenn Sie diese Warnung ignorieren und mit *Accept the Risk and Continue* die Verbindung dennoch aufbauen würden (tun Sie das hier aber nicht)?

Man riskiert, dass man mit einem Webserver verbunden ist, dass nicht zertifiziert ist und somit potenzielle Daten wie Name, E-Mail und Passwörter abfangen kann.

Damit der Browser das Zertifikat des Webservers akzeptiert, muss also noch das CA-Zertifikat als vertrauenswürdiges Zertifikat im Browser installiert werden. Kopieren Sie dazu zuerst das CA-Zertifikat nach */user/home* und ändern Sie den Owner des Files auf *user* (machen Sie dies als *root*):

```
$ cp /etc/ssl/cacerts/caCert.pem /home/user/
$ chown user:user /home/user/caCert.pem
```

Wählen Sie dann in Firefox *Menü-Button oben rechts* → *Settings* → dann links *Privacy & Security* und scrollen Sie zu *Certificates*. Klicken Sie auf den Button *View Certificates* und dort im Tab *Authorities* auf *Import*. Selektieren Sie das CA-Zertifikat, das sich in */home/user/caCert.pem* befindet, und klicken Sie *Open*.

Im nächsten Fenster geben Sie die Trust-Settings für das CA-Zertifikat an. Wählen Sie hier *Trust this CA to identify websites* aus, damit von diesem Zertifikat ausgestellte Server-Zertifikate als vertrauenswürdig akzeptiert werden. Anschliessend sollte das Zertifikat in der Liste der vertrauenswürdigen Zertifikate auftauchen (ganz unten).

Der Schritt, den Sie hier gemacht haben, ist sehr signifikant: Ihr CA-Zertifikat wird nun vom Browser genau gleich behandelt wie die bereits vorinstallierten und damit «offiziell vertrauenswürdigen» Zertifikate. Nehmen wir an, Sie machen so etwas in Ihrer eigenen Firma mit 500 Mitarbeitern, d.h. Sie betreiben firmenintern eine eigene CA und installieren das CA-Zertifikat in allen Browsern auf den Firmenrechnern der Mitarbeiter. Beeinflusst dies, wie sicher sich Ihre Mitarbeiter sein können, dass wenn diese z.B. *https://www.google.ch* kontaktieren und keine Zertifikatswarnung erscheint, sie auch wirklich direkt mit *www.google.ch* verbunden sind und niemand mithört?

....

Nehmen Sie nun an, ein Mitarbeiter gelangt an den Private Key der CA. Was kann er damit nun tun?

Alle Verbindungen mit dem CA können nun entschlüsselt werden.

Es ist also von grosser Wichtigkeit, dass Sie bei einem solchen Szenario die Sicherheit sehr ernst nehmen und den Private Key Ihrer eigenen CA sehr gut schützen.

Schliessen Sie nun den Browser, starten Sie ihn neu und greifen Sie wiederum auf den Webserver zu. Jetzt sollte alles funktionieren (ohne Warnung) und Sie erhalten einen entsprechenden Text im Browser.

Klicken Sie auf das Schlosssymbol links neben der Adresszeile, dann auf *More Information* und schliesslich auf den Button *View Certificate*. Dort sehen Sie oben die ganze Certificate-Chain[3]: Das Zertifikat von *host.securitylab.local* wurde von der CA mit Namen *GruppeX_CA* ausgestellt. Auch hier können Sie die ganzen Details in den Zertifikaten anschauen, dabei sehen Sie genau dasselbe wie im vorhergehenden Abschnitt mit *openssl*.

Beenden und starten Sie den Browser erneut. Kontaktieren Sie den Webserver dann über dessen IP-Adresse statt den Hostnamen (natürlich müssen Sie hier die richtige IP-Adresse Ihrer eigenen virtuellen Maschine verwenden):

```
https://192.168.57.131
```

Erstaunlicherweise erhalten Sie nun wieder eine Warnung, obwohl das CA-Zertifikat korrekt installiert ist und obwohl Sie mit dem richtigen Webserver kommunizieren. Was ist hier das Problem?

Zertifikat wurde auf den Hostname gewählt, aber nicht auf die IP-Adresse

Betrachten Sie nochmals das Zertifikat des Webservers (via Schlosssymbol in der Adresszeile). Welcher Eintrag im Zertifikat wird vom Webbrowser verwendet, um festzustellen, zu welchem Webserver (bzw. zu welchem Hostnamen) das Zertifikat genau gehört? Seien Sie hier präzise: Wenn es sich nur um einen Teil eines der Felder im Zertifikat handelt, dann bezeichnen Sie dies genau.

Im "Subject Name" und danach "Common Name"

Sie haben sie vermutlich etwas schwer getan bei der Beantwortung dieser Frage, dann *host.securitylab.local* kommt sowohl im Subject (*Common Name*) als auch bei den *Subject Alt Names* vor. In der Anfangszeit von Zertifikaten gab es die *Subject Alt Names* Extension noch nicht und entsprechend wurde damals immer der *Common Name* verwendet, um festzustellen, zu welchem Webserver das Zertifikat gehört. Die Einführung der *Subject Alt Names* Extension hat dies flexibilisiert, weil dort auch mehrere Hostnamen eingetragen werden können. Heute ist es so, dass wenn im Zertifikat die Extension *Subject Alt Name* vorhanden ist, dann wird bei der Überprüfung der Zugehörigkeit des Zertifikats immer diese Extension verwendet, und der *Common Name* wird ignoriert. Im vorliegenden Fall wurde also der in *Subject Alt Names* eingetragene Hostname verwendet. Zudem ist es so, dass heute einige der Browser den *Common Name* überhaupt nicht mehr auswerten, sie verlangen also in jedem Fall die *Subject Alt Names* Extension. Firefox macht dies z.B. seit Ende 2021 so.

3.5 Rekapitulation: Überprüfung eines Zertifikats

Basierend auf den obigen Erklärungen und Ihren Antworten auf die Fragen sollten Sie nun gut verstehen, was genau passiert, wenn der Browser ein Zertifikat von einem Server erhält und dieses prüft. Für

[3] Die *Certificate Chain* bezeichnet die Zertifikatskette vom self-signed CA-Zertifikat, über allfällige Intermediate Zertifikate, bis zum Endzertifikat (z.B. das eines Servers). In unserem Fall besteht die Chain aus 2 Zertifikaten, dem CA-Zertifikat und dem Zertifikat des Webservers.

die Internetsicherheit ist dies ein fundamental wichtiger Ablauf, den Sie genau verstehen sollten. Im Folgenden geht es deshalb darum, diese Schritte nochmals genau aufzuführen und in diesem Zusammenhang auch um die Details, die in jedem Schritt durchgeführt werden.

Wir beziehen uns dabei auf unseren Fall mit einem CA-Zertifikat und einem direkt damit ausgestellten Zertifikat des Webservers. Details wie OCSP-Abfragen oder Certificate-Revocation-Lists werden ignoriert, da dies in unserem Fall nicht verwendet wird.

Die groben Schritte sind dabei vorgegeben; Ihre Aufgabe ist es, genau zu beschreiben, was in den einzelnen Schritten im Detail gemacht wird. Am Schluss soll der Browser «überzeugt» sein, dass er das richtige Zertifikat erhalten hat (das des Servers, mit dem er kommunizieren will) und dass dieses vertrauenswürdig ist.

Die Ausgangslage sei wie folgt:

- Der Browser initiiert eine TLS-Verbindung mit *host.securitylab.local*
- Der Browser erhält dabei ein Zertifikat des Webservers

Im nächsten Schritt sucht der Browser das richtige CA-Zertifikat unter den vorinstallierten CA-Zertifikaten, d.h. das Zertifikat der CA, die das Webserver-Zertifikat ausgestellt hat. Welche Teilschritte werden dabei durchgeführt? Geben Sie hier insbesondere auch die Felder in den Zertifikaten ein, die verwendet werden, um das richtige CA-Zertifikat zu finden.

Vorinstallierte CA-Zertifikate werden auf eine Übereinstimmung überprüft

preinstalled->subject = server->issuer

Ist das CA-Zertifikat gefunden so geht es im nächsten Schritt darum zu überprüfen, ob das Webserver-Zertifikat tatsächlich von der CA ausgestellt (signiert) wurde. Welche Teilschritte werden dabei durchgeführt? Gehen Sie hier insbesondere darauf ein, welche Schlüssel wozu verwendet werden.

Der Browser entschlüsselt die Signatur des Webserver-Zertifikats mit dem übereinstimmenden CA-Public-Key. Das Resultat sollte mit dem Webserver-Zertifikat übereinstimmen.

Jetzt weiss der Browser, dass er ein gültiges, von einer offiziellen CA ausgestelltes Server-Zertifikat hat. Nun prüft der Browser, ob das Zertifikat zeitlich gültig ist und ob es wirklich dem Server gehört, mit dem der Browser kommunizieren möchte. Welche Teilschritte werden dabei durchgeführt?

Die "validity" vom Zertifikat wird überprüft.

Damit ist die Überprüfung des Zertifikats komplett: Der Browser weiss nun, dass er ein gültiges, von einer vertrauenswürdigen CA ausgestelltes Zertifikat erhalten hat und zudem auch, dass dieses zum

Server gehört, mit dem der Browser kommunizieren möchte. Damit kennt der Browser nun auch den Public Key des Servers.

Ganz wichtig: Damit ist nur das Zertifikat überprüft, aber der Server ist noch nicht authentisiert – denn jeder Server kann dem Browser grundsätzlich ein beliebiges Zertifikat schicken (diese sind ja öffentlich und man kann einfach das Zertifikat jedes Servers erhalten, indem man eine TLS-Verbindung mit ihm aufbaut). Das Einzige, was wir bisher erreicht haben ist, dass der Browser nun den richtigen Public Key des Servers kennt, mit dem er kommunizieren möchte. Um den Server nun auch noch zu authentisieren, muss der Server beweisen, dass er den zugehörigen Private Key kennt; z.B. indem er eine Signatur leistet über einem vom Browser gewählten Wert. Diese Signatur kann dann vom Browser mit dem Public Key aus dem Webserver-Zertifikat überprüft werden – und dann ist der Server auch wirklich authentisiert. Sie werden die später noch im Detail analysieren.

3.6 Benutzer-Authentisierung im Webserver konfigurieren

Bisher haben wir eine einseitige Authentisierung der TLS-Verbindung: Der Browser weiss, dass er mit dem richtigen Server kommuniziert, aber der Webserver weiss noch nichts über den Benutzer. In der Praxis wird dies auch meist so gehandhabt. Wenn zusätzlich eine Benutzer-Authentisierung erwünscht ist, so wird dies meist nach dem Aufbau der TLS-Verbindung als Teil der Applikation gemacht (z.B. mit einem Login-Formular bei einer Webapplikation).

Wie Sie wissen kann sich der Benutzer bzw. der Client aber auch als Teil von TLS authentisieren. Dazu müssen wir zuerst einige Anpassungen in der Konfiguration des Servers vornehmen. Öffnen Sie dazu */etc/apache2/sites-available/lab_certificates.conf* mit einem Texteditor und lesen Sie den Kommentar vor dem Eintrag *SSLVerifyClient*.

Wie Sie sehen wird damit spezifiziert, ob sich der Client auch authentisieren muss. Derzeit ist das nicht der Fall, deshalb ist die aktuelle Konfiguration *none*. Um Client-Authentisierung zu verlangen, ändern Sie den Eintrag zu

```
SSLVerifyClient require
```

Bei der Client-Authentisierung mit TLS teilt der Server dem Client mit, von welchen CAs der Server-Zertifikate akzeptiert (siehe Vorlesungsunterlagen). In unserem Fall soll der Server nur Benutzer-Zertifikate akzeptieren, die von Ihrer eigenen CA ausgestellt wurden. Dazu ist der Eintrag *SSLCACertificateFile* (derzeit kommentiert) wichtig. Lesen Sie auch hier den Kommentar vor diesem Eintrag durch.

Um dies richtig zu konfigurieren, müssen Sie also das Kommentarzeichen vor *SSLCACertificateFile* entfernen und zusätzlich das CA-Zertifikat an die dort angegebene Location kopieren (**Achtung: *Kopieren*, nicht verschieben, d.h. verwenden Sie *cp* und nicht *mv*!**). Sie finden das CA-Zertifikat unter */etc/ssl/cacerts/caCert.pem*.

Starten sie danach den Webserver neu. Schliessen Sie ebenfalls den Webbrowser und starten Sie diesen erneut und versuchen Sie wiederum, eine TLS-Verbindung vom Browser zum Server aufzusetzen. Was beobachten Sie? Was ist passiert? Wieso funktioniert das nicht? Die Fehlermeldung erlaubt nicht wirklich einen klaren Rückschluss auf das dahinterliegende Problem; weitere Informationen erhalten Sie auch aus dem Error-Logfile von Apache (*/var/log/apache2/error.log*).

vlt. brauchen wir noch User Zertifikate?

```
[Wed Dec 07 13:32:58.402592 2022] [ssl:info] [pid 6511:tid 139737522075200] [client 192.168.142.128:54820] AH01964: C
onnection to child 67 established (server host.securitylab.local:443)
[Wed Dec 07 13:32:58.405887 2022] [ssl:info] [pid 6511:tid 139737522075200] [client 192.168.142.128:54820] AH02008: S
SL library error 1 in handshake (server host.securitylab.local:443)
[Wed Dec 07 13:32:58.405916 2022] [ssl:info] [pid 6511:tid 139737522075200] SSL Library Error: error:0A0000C7:SSL rou
tines::peer did not return a certificate -- No CAs known to server for verification?
[Wed Dec 07 13:32:58.405922 2022] [ssl:info] [pid 6511:tid 139737522075200] [client 192.168.142.128:54820] AH01998: C
onnection closed to child 67 with abortive shutdown (server host.securitylab.local:443)
```

3.7 Schlüsselpaar und Zertifikat für den Benutzer erzeugen

Offensichtlich brauchen wir auch noch ein Schlüsselpaar und ein Zertifikat für den Benutzer bzw. den Client. Das funktioniert genau gleich wie zuvor mit dem Webserver-Zertifikat.

Erzeugen Sie zuerst ein Schlüsselpaar und einen zugehörigen Zertifikats-Request:

```
$ openssl req -newkey rsa:2048 -keyout userKey.pem -out userReq.pem
```

Der Private Key wird in *userKey.pem* und der Zertifikats-Request in *userReq.pem* abgespeichert.

Nachdem Sie den Befehl ausgeführt haben, müssen Sie diverse Informationen angeben:

- Eine Passphrase, mit der der Private Key geschützt (verschlüsselt) wird. Wählen Sie hier die Passphrase *user-passphrase*, um diese eindeutig von anderen Passphrases zu unterscheiden.
- Informationen zur Identität (X.500 Name) des Benutzers. Geben Sie hier bei *Country Name* den Wert *CH* ein, bei *Organization Name* den Wert *ZHAW* und bei *Common Name* den Wert *GruppeX_User*, wobei *X* Ihrer Gruppennummer entspricht. *E-Mail Address* können Sie leer lassen.

Im Gegensatz zum Zertifikat des Webservers muss hier der *Common Name* nicht einem spezifischen Wert wie dem Hostnamen des Webservers entsprechen und effektiv wird der *Common Name* im Benutzer-Zertifikat vom Server beim TLS-Verbindungsaufbau auch gar nicht geprüft. Es ist die Aufgabe des Webservers bzw. der Webapplikation, den *Common Name* im Zertifikat auszuwerten und entsprechende Zugriffsrechte zu erteilen. Aus der Sicht von TLS ist das out-of-scope und deshalb für dieses Praktikum nicht relevant.

3.8 Zertifikat für den Benutzer ausstellen

Der eben erzeugte Zertifikats-Request wird nun der CA zum Ausstellen des Zertifikats übergeben.

Bevor Sie das Zertifikat ausstellen, müssen Sie eine Anpassung in */etc/ssl/openssl.cnf* vornehmen. Editieren Sie das File und fügen Sie bei der Zeile *subjectAltName=@alt_names* zu Beginn ein Kommentarzeichen hinzu (#), damit im Gegensatz zum Zertifikat für den Webserver keine *Subject Alt Names* Extension verwendet wird.

Nun nehmen Sie wieder die Rolle der CA ein und erzeugen aus dem Zertifikats-Request das Zertifikat für den Benutzer. Dies machen Sie analog zum Ausstellen des Webserver-Zertifikats mit folgendem Befehl:

```
$ openssl ca -in userReq.pem -days 365 -out userCert.pem
```

Nach Eingabe des Befehls müssen Sie die Passphrase (*ca-passphrase*) eingeben, mit welchem der Private Key der CA geschützt ist.

Wenn Sie möchten, so können Sie den obigen Request und das ausgestellte Zertifikat mit dem passenden *openssl*-Befehl anschauen, die Inhalte sehen aber grundsätzlich (bis auf den *Common Name* und *Subject Alt Names*) gleich aus wie beim Request und dem Zertifikat des Webservers, Sie werden dabei also nichts Neues lernen.

3.9 Private Key und Zertifikat des Benutzers im Browser installieren

Schliesslich müssen wir Private Key und Zertifikat noch im Browser installieren. Dies macht man, indem man Private Key und Zertifikat in ein PKCS #12-File einfügt. PKCS #12 ist ein Standard Fileformat, um Private Keys zusammen mit Zertifikaten abzuspeichern. Verwenden Sie dazu folgenden Befehl:

```
$ openssl pkcs12 -export -inkey userKey.pem -in userCert.pem
  -out /home/user/userCert.p12
```

Da hier auf den Private Key des Benutzers zugegriffen wird, müssen Sie nach Eingabe des Befehls die Passphrase (*user-passphrase*) eingeben. Anschliessend werden Sie noch nach einem Export-Passwort gefragt, um den Private Key im PKCS #12-File zu schützen. Verwenden Sie hier *p12-passphrase*. Das erzeugte File hat den Namen *userCert.p12* und befindet sich unter */home/user/*. Ändern Sie nun noch den Owner des Files auf *user* (machen Sie dies als *root*):

```
$ chown user:user /home/user/userCert.p12
```

Importieren Sie nun das Zertifikat und den Private Key im Firefox-Browser. Wählen Sie in Firefox *Menü-Button oben rechts* → *Settings* → dann links *Privacy & Security* und scrollen Sie zu *Certificates*. Klicken Sie auf den Button *View Certificates* und dort im Tab *Your Certificates* auf *Import*. Selektieren Sie das zuvor exportierte PKCS #12 File (*/home/user/userCert.p12*), dabei müssen Sie wieder die Passphrase (*p12-passphrase*) eingeben. Jetzt sehen Sie das importierte Benutzer-Zertifikat aufgelistet. Wählen Sie es an und klicken Sie auf *View*. Hier sehen Sie wiederum die Detailinformationen des Zertifikats.

Um zu testen, ob alles funktioniert, schliessen Sie den Browser, öffnen ihn wieder und starten eine TLS-Verbindung um Server. Sie erhalten einen Dialog, der Sie informiert, dass Sie sich mit einem Zertifikat authentisieren müssen (falls Sie den Dialog nicht sehen, klicken Sie den *Reload* button). Das einzige vorhandene Zertifikat ist bereits ausgewählt, bestätigen Sie dessen Verwendung mit einem Klick auf den *OK*-Button. Wenn die TLS-Verbindung steht, erhalten Sie durch einen Klick auf das Schlosssymbol links neben der Adresszeile und der Auswahl von *More Information* weitere Informationen über die geschützte Verbindung. Informationen, dass sich der Client auch authentisiert hat, suchen Sie jedoch vergebens – ein Zeichen dafür, dass die Browser Client-Authentisierung zwar unterstützen, diese in der Praxis derzeit aber nur eine geringe Rolle spielt.

4 Analyse des TLS-Handshake mit Wireshark

In diesem Teil werden Sie den eben durchgeführten TLS-Verbindungsaufbau genauer analysieren, um nachzuvollziehen, was dabei im Detail passiert ist. Dies wird Ihnen helfen, das TLS-Protokoll und die Verwendung der Zertifikate besser verstehen. Die Analyse werden Sie mit *Wireshark* durchführen. Damit *Wireshark* problemlos läuft, sollten Sie es in einem Terminal als *user* durch Eingabe von *sudo wireshark* (gefolgt vom Passwort *user*) starten. Starten Sie dann den Browser nochmals neu.

Um den TLS-Handshake aufzuzeichnen, müssen Sie im *Capture Interfaces*-Fenster (erreichbar über den Menüpunkt *Capture* → *Options*) das Loopback-Interface (*lo*) auswählen, da wir ja eine rein lokale Kommunikation aufzeichnen. Das eigentliche Aufzeichnen beginnt nach einem Klick auf den Button *Start*.

Da TLS 1.3 verwendet wird, wird ein substantieller Teil des TLS-Handshake verschlüsselt durchgeführt. *Wireshark* kann dies und auch den nachfolgenden Datenaustausch aber entschlüsseln, wenn man das verwendete Schlüsselmaterial in *Wireshark* konfiguriert. Dazu müssen Sie zuerst an das Schlüsselmaterial gelangen. Dies kann Ihnen Firefox liefern. Löschen Sie dazu zuerst in Firefox den Cache, damit beim nächsten Start sicher eine neue TLS-Session aufgebaut wird (*Menü-Button oben rechts* → *Settings* → *Privacy & Security* → *Clear History Button* → *alle Checkboxen auswählen* → *OK*). Schliessen Sie dann Firefox und starten Sie Firefox anschliessend in einem Terminal (als *user*) wie folgt:

```
$ SSLKEYLOGFILE="key.txt" firefox
```

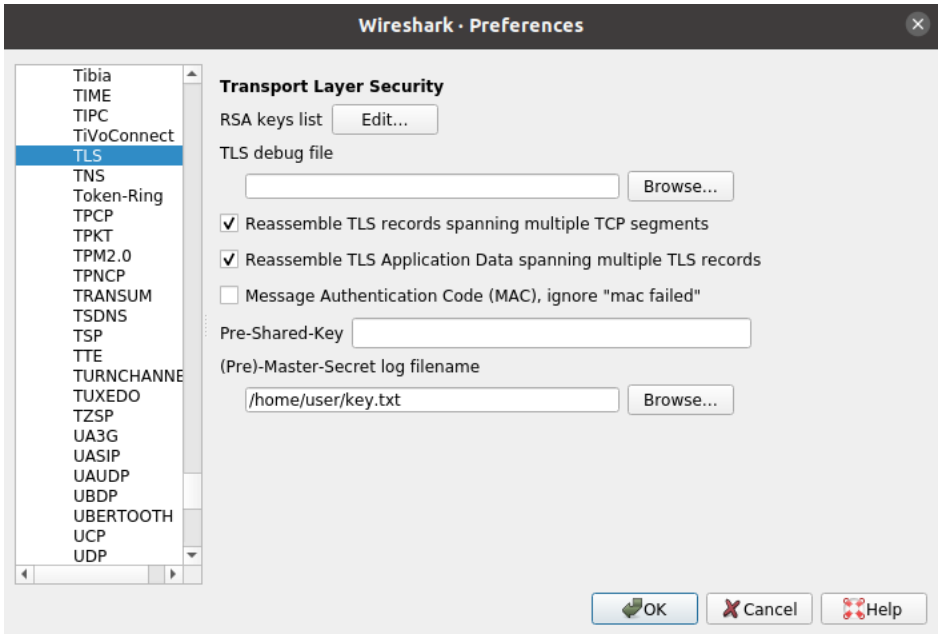
Durch die Verwendung der Umgebungsvariablen *SSLKEYLOGFILE* werden die für die Schlüsselgenerierung relevanten Informationen im angegebenen File – hier *key.txt* – abgespeichert. Verbinden Sie sich im Browser nun mit *https://host.securitylab.local*, schliessen Sie den Browser und stoppen Sie dann die Aufzeichnung in *Wireshark* mit dem Menüpunkt *Capture* → *Stop*. Geben Sie am besten oben im Filter-Feld von *Wirehark* noch *tcp.port==443* ein, damit keine störenden Pakete sichtbar sind. Wenn Sie den Handshake nun in *Wireshark* anschauen, dann werden Sie nur ein paar wenige TLS-

Pakete im Plaintext sehen und ab einem gewissen Punkt nur noch *Application Data* – weil ab da die Pakete verschlüsselt werden.

Betrachten Sie den Inhalt des Files *key.txt*. Dieser sollte in etwa wie folgt aussehen:

```
CLIENT_HANDSHAKE_TRAFFIC_SECRET b53935881ad46e3b988e0303fcb44...
SERVER_HANDSHAKE_TRAFFIC_SECRET b53935881ad46e3b988e0303fcb44...
...
CLIENT_TRAFFIC_SECRET_0 1fbe5deb08c7d38feabfa9af2b95483ff36ce...
SERVER_TRAFFIC_SECRET_0 1fbe5deb08c7d38feabfa9af2b95483ff36ce...
...
EXPORTER_SECRET 1fbe5deb08c7d38feabfa9af2b95483ff36cee66cfeea...
...
```

Das sind im Wesentlichen alle sicherheitsrelevanten Daten, die während des Handshake ausgetauscht werden und mit denen das ausgehandelte Schlüsselmaterial erzeugt werden kann. Wählen Sie nun in Wireshark *Menü Edit → Preferences* und spezifizieren Sie beim TLS Protokoll das File *key.txt*, wie nachfolgend dargestellt:



Nun sollte *Wireshark* sowohl den TLS-Handshake als auch den nachfolgenden Datenaustausch im Plaintext darstellen. Analysieren Sie nun den Handshake und beantworten Sie folgende Fragen. Verwenden Sie dabei zur Unterstützung auch die Unterlagen aus der Vorlesung.

Analysieren Sie die Liste der vom Client angebotenen Cipher Suites (in *Client Hello*). Wieviele Cipher Suites werden angeboten? Erkennen Sie hier Cipher Suites, die heute nicht mehr verwendet werden sollten?

nicht sicher...vlt. CBC?

```
▾ Cipher Suites (17 suites)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
    Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
```

Welche Cipher Suite wird vom Server ausgewählt (in *Server Hello*)?

Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)

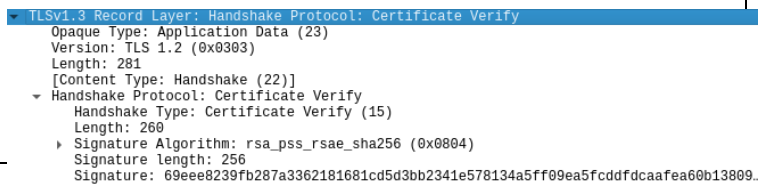
Nach der *Server Hello*-Nachricht sendet der Server unter anderem eine *Certificate*-Nachricht. Darin sendet der Server sein Zertifikat an den Browser. Wenn Sie die Nachricht genau anschauen, dann sehen Sie, dass der Server auch das CA-Zertifikat mitsendet. Apache verhält sich so, weil wir das CA-Zertifikat ebenfalls im Verzeichnis */etc/apache2/ssl.crt* installiert haben – es werden dann einfach alle dort enthaltenen Zertifikate mitgesendet. Der Browser wird dieses CA-Zertifikat aber ignorieren und den Bezug des Server-Zertifikats zu den im Browser vorkonfigurierten, vertrauenswürdigen Zertifikaten herstellen.

Dass der Webserver sein eigenes Zertifikat mitsendet und dieses vom Browser korrekt verifiziert wird ist eine wichtige und notwendige Grundlage dafür, dass sich der Webserver gegenüber dem Browser authentisieren kann. Wie bereits vorgehend erwähnt, reicht dies aber natürlich nicht, weil ein erhaltenes Zertifikat alleine noch nichts darüber aussagt, mit wem man wirklich kommuniziert. Für die eigentliche Authentisierung muss der Webserver dem Browser deshalb auch noch beweisen, dass er den zugehörigen Private Key besitzt.

Wie beweist der Webserver dem Browser im vorliegenden Fall, dass er den Private Key besitzt? Und in welcher Handshake-Nachricht befindet sich die zugehörige Information?

Hinweis: Am besten konsultieren Sie erst die Vorlesungsunterlagen, um herauszufinden, wie das funktioniert. Sobald Sie anhand der Vorlesungsunterlagen verstanden haben, wie das funktioniert, sollte es möglich sein, die relevanten Informationen auch im aufgezeichneten TLS-Handshake zu finden.

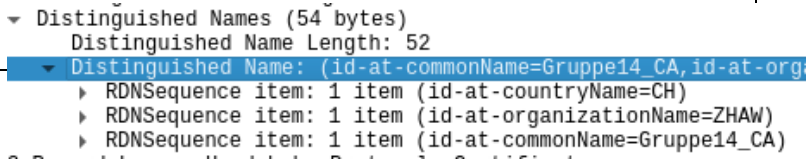
Webserver sendet eine Signature, die mit dem Private Key erstellt wurde, an den Browser, welcher dann die Signatur mit dem Public Key entschlüsselt...



Dass sich zusätzlich auch der Client beim Server authentisiert, erkennt man an drei zusätzlichen Nachrichten im TLS-Handshake, die bei rein serverseitiger Authentisierung nicht verwendet werden: *CertificateRequest* (vom Server zum Client), *Certificate* (vom Client zum Server) und *CertificateVerify* (vom Client zum Server). Studieren Sie dazu erst die Vorlesungsunterlagen, damit Sie die Semantik der Nachrichten verstehen und analysieren Sie diese ebenfalls im aufgezeichneten Handshake. Beschreiben Sie dann den Zweck der drei Nachrichten.

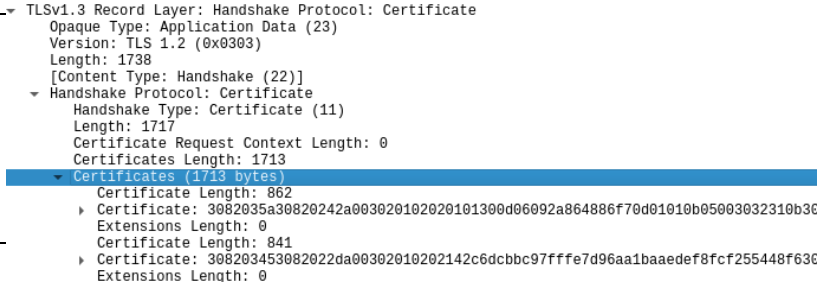
CertificateRequest (gehen Sie hier insb. auch auf den in dieser Nachricht mitgesendeten *Distinguished Name* ein):

Server verlangt vom Client, dass er sich mit einem Zertifikat authentisieren soll. Zusätzlich gibt der Server eine Liste von möglichen Hash Varianten für die Signatur vor.
Server gibt auch schon vor wer der Issuer sein soll sprich Gruppe14_CA.



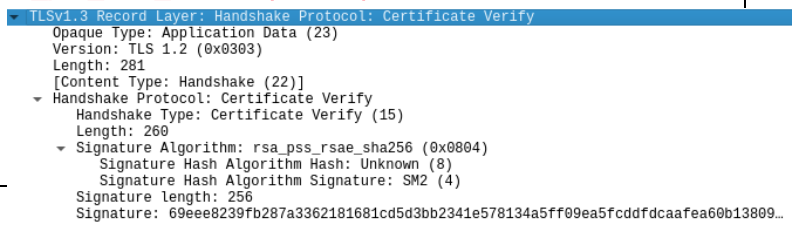
Certificate:

Client übergibt dem Server sein Zertifikat.



CertificateVerify:

Client wählt Signature Algorithm: rsa_pss_rsae_sha256 (0x0804) und schickt noch die Signatur



Insbesondere die letzten beiden Nachrichten erlauben es dem Server, den Client zu authentisieren, da diese beiden Nachrichten einerseits das Zertifikat des Clients liefern (das vom Webserver natürlich überprüft wird, ähnlich wie in Abschnitt 3.5) und andererseits den Beweis beinhalten, dass der Client den zugehörigen Private Key kennt (mittels einer Signatur, die vom Server mit Verwendung des Public Keys im erhaltenen Zertifikat verifiziert werden kann).

5 Weitere Betrachtungen zu Zertifikaten

Im letzten Teil wird auf einige weitere Aspekte von Zertifikaten eingegangen, die wir bisher noch nicht untersucht haben. Dabei arbeiten Sie nicht mehr mit den eigenen Zertifikaten, sondern untersuchen die Interaktion mit öffentlich verfügbaren Servern.

5.1 Längere Certificate-Chains

Bisher bestand die Certificate-Chain immer aus zwei Zertifikaten: dem CA-Zertifikat und dem Zertifikat des Webservers. Öffentliche CAs verwenden heute typischerweise eine oder teilweise auch zwei Intermediate Zertifikate, siehe dazu die Vorlesungsunterlagen. Im Folgenden analysieren Sie, wie dabei die Chain gebildet wird.

Als Beispiel verwenden wir *https://www.zhaw.ch*. Starten Sie den Browser neu (wiederum in einem Terminal mit Angabe der Umgebungsvariablen *SSLKEYLOGFILE*) und verwenden Sie *Wireshark*, um den Verbindungsaufbau aufzuzeichnen. Dieses Mal müssen Sie das externe Interface verwenden (vermutlich *ens33* oder *eth0*). Kontaktieren Sie dann mit dem Browser *https://www.zhaw.ch*.

Betrachten Sie nun im Browser die Certificate-Chain (wiederum über das Schlosssymbol links neben der Adresszeile). Die Chain besteht nun aus drei Zertifikaten: Aus dem self-signed Root-CA-Zertifikat, einem Intermediate-Zertifikat der CA und dem Zertifikat von *www.zhaw.ch*. Konkret wurde mit dem Root-CA-Zertifikat das Intermediate-CA-Zertifikat ausgestellt (d.h. mit dem entsprechenden Private Key signiert) und mit dem Intermediate-CA-Zertifikat wurde das Webserver-Zertifikat ausgestellt.

Damit die komplette Chain geprüft werden kann muss der Browser alle drei Zertifikate kennen. Stellen Sie eine Vermutung auf, wie der Browser während dem TLS-Handshake zu allen drei Zertifikaten gelangt. Analysieren Sie den aufgezeichneten Handshake noch nicht.

Jeder CA wird abgefragt. Zuerst den Intermediate und dann den Root-CA.

Analysieren Sie nun die *Certificate*-Nachricht des TLS-Handshake in *Wireshark* (wird vom Webserver zum Browser gesendet). Wieviele und welche Zertifikate sind dort erhalten? Das mittlere Fenster von *Wireshark* listet die Details im Zertifikat schön lesbar auf; damit können Sie die Zertifikate gut identifizieren.

Basierend auf diesen neuen Erkenntnissen: Müssen Sie Ihre Antwort von vorhin («wie gelangt der Browser zu allen drei Zertifikaten») revidieren?

Ich glaube, der Root-CA ist schon vorinstalliert und deshalb wird nur der Intermediate CA noch zusätzlich verlangt und überprüft.

Im Fall von *www.zhaw.ch* wird offensichtlich die komplette Chain an Zertifikaten an den Browser gesendet, das Root-CA-Zertifikat wäre aber nicht nötig (und wird auch nicht von allen Webservern mitgesendet). Wichtig sind nur das Intermediate-CA-Zertifikat und das Zertifikat des Webservers. Damit diese zum Browser gesendet werden können, müssen also in einem Webserver nicht nur das Webserver-Zertifikat (und der zugehörige Private Key) installiert werden, sondern zusätzlich auch das Intermediate-CA-Zertifikat (wenn ein solches verwendet wird), da dieses in der Regel nicht im Browser vorinstalliert ist.

Nun sollten Sie verstehen, wie eine Certificate-Chain der Länge 3 (oder auch der Länge n) verifiziert werden kann. Die Vorgehensweise ist grundsätzlich gleich wie mit zwei Zertifikaten (siehe Abschnitt 3.5), es muss nun einfach noch ein weiteres Zertifikat geprüft werden. Die wichtigsten Schritte für eine Certificate-Chain der Länge 3 sind im Folgenden angegeben:

- Der Browser sucht sich alle Zertifikate der Chain zusammen. Im Normalfall befindet sich das Root-CA-Zertifikate unter den vertrauenswürdigen, im Browser gespeicherten CA-Zertifikate und die anderen Zertifikate (typischerweise Intermediate-CA-Zertifikat und Zertifikat des Webservers) werden vom Webserver erhalten. Das Matching wird dabei in gewohnter Manier mit den Feldern *Issuer* bzw. *Subject* in den Zertifikaten gemacht.
- Nun wird zuerst das Intermediate-CA-Zertifikat geprüft. Dabei wird der Public Key aus dem Root-CA-Zertifikat verwendet, um die Signatur im Intermediate-CA-Zertifikat zu verifizieren. Kann diese erfolgreich verifiziert werden, so weiss der Browser, dass das Intermediate-CA-Zertifikat von der CA ausgestellt wurde, wodurch der Browser dem Intermediate-CA-Zertifikat und damit dem im Zertifikat erhaltenen Public Key ebenfalls vertrauen kann.

- Dieser Public Key wird dann verwendet, um die Signatur im Webserver-Zertifikat zu verifizieren. Kann auch dies erfolgreich gemacht werden, so weiss der Browser, dass er nun ein gültiges und damit vertrauenswürdiges Zertifikat des Webservers hat.
- Alles andere (Überprüfung Gültigkeitsdauer, Überprüfung *Subject Alt Names* (oder *Common Name*) im Webserver-Zertifikat etc.) wird genau gleich gemacht wie bei einer Certificate-Chain der Länge 2.

Im Vergleich zu einer Certificate-Chain der Länge 2 muss also einfach ein weiteres Zertifikat überprüft werden und der Ansatz lässt sich entsprechend auch auf längere Chains erweitern. Entscheidend dabei ist, dass der Browser nur dem Root-CA-Zertifikat vertrauen muss und entsprechend auch nur dieses im Browser als vertrauenswürdig installiert sein muss. Dieses Root-CA-Zertifikat gibt dann das Vertrauen einfach «über mehrere Zertifikate» weiter.

Nebenbei: Browser cachen teilweise die von Webservern erhaltenen Intermediate-CA-Zertifikate, auch wenn dies grundsätzlich nicht notwendig wäre. Das können Sie gleich selbst verifizieren: Wenn Sie sich jetzt die in Firefox installierten CA-Zertifikate betrachten, dann werden Sie dort sowohl das Root-CA-Zertifikat als auch das Intermediate CA-Zertifikat von *www.zhaw.ch* finden. Das Root-CA-Zertifikat ist dabei mit *Builtin Object Token* bezeichnet, weil es vorinstalliert ist, das Intermediate-CA-Zertifikat «nur» mit *Software Security Device*, weil es ein gecachtes Zertifikat ist.

Und nochmals nebenbei: Wenn Sie das Zertifikat von *www.zhaw.ch* nochmals studieren, dann werden Sie sehen, dass die *Subject Alt Names* Extension eine Vielzahl von Einträgen enthält. Das Zertifikat kann also nicht nur für *www.zhaw.ch* verwendet werden (über den Eintrag **.zhaw.ch* – ja, auch solche Wildcard-Hostnamen sind möglich), sondern auch für weitere Hostnamen.

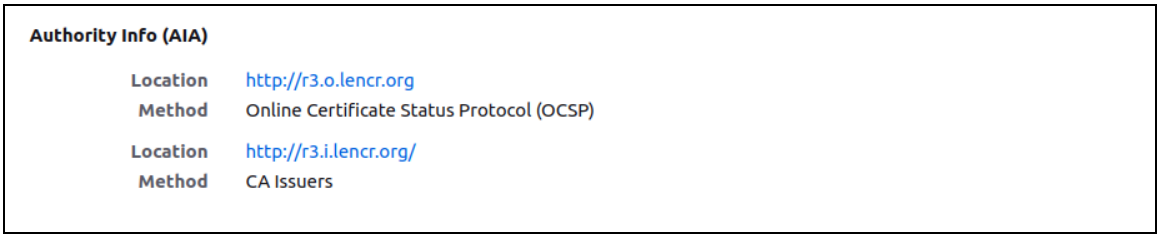
5.2 OCSP

Wie Sie in der Vorlesung erfahren haben. genügt es nicht, wenn nur das Zertifikat selbst überprüft wird. Zusätzlich muss bei der CA nachgefragt werden, ob das Zertifikat noch gültig ist, denn es könnte in der Zwischenzeit revoziert worden sein, z.B. weil der zugehörige Private Key kompromittiert wurde. Dazu wird heute üblicherweise OCSP[4] verwendet. In diesem und den folgenden beiden Abschnitten werden Sie OCSP etwas genauer unter die Lupe nehmen.

Um die erste Analyse durchzuführen, verwenden Sie wiederum *https://www.zhaw.ch*. Sie können entweder den zuvor aufgezeichneten Handshake verwenden oder auch einen neuen durchführen.

Eine OCSP-Abfrage besteht aus einem Request und einer Response und wird über HTTP (port 80) gesendet. Dazu muss der Browser zuerst einmal wissen, an wen er den OCSP/HTTP-Request richten sollte (d.h. den OCSP-Server, der oft auch als OCSP-Responder oder OCSP-Endpoint bezeichnet wird). Woher weiss das der Browser?

Hinweis: Schauen Sie das Zertifikat von *www.zhaw.ch* genauer an.

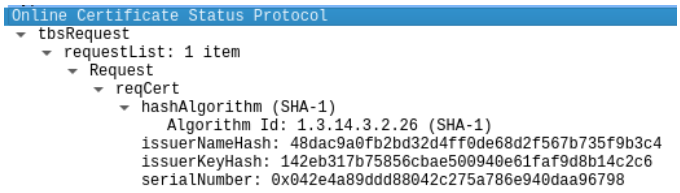


Suchen Sie in Wireshark nun den zugehörigen OCSP-Request und die Response. Diese sollten kurz nach dem TLS-Handshake sichtbar sein (im Protocol-Feld steht *OCSP*). Wenn Sie eine der Nachrichten auswählen, so sehen Sie im mittleren Fenster, dass die OCSP-Nachrichten wie bereits erklärt über HTTP gesendet werden.

[4] https://tools.ietf.org/html/rfc6960

Betrachten Sie zuerst den OCSP-Request und beachten Sie die verschiedenen Informationen, die dabei zum OCSP-Responder gesendet werden. Dabei ist insbesondere ein Feld wichtig, damit der OCSP-Responder weiss, von welchem Zertifikat der Client den Status abfragen möchte. Welches Feld ist das und welches Zertifikat in der Certificate-Chain wird damit identifiziert? Um das zu beantworten, müssen Sie zusätzlich auch die Details der Zertifikate der Certificate-Chain anschauen.

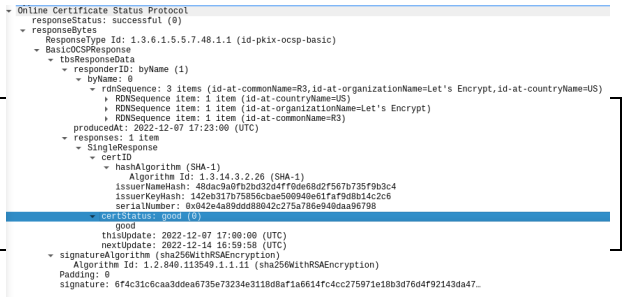
Name ist gehashet.



Betrachten Sie nun die OCSP-Response. Diese ist komplexer aufgebaut als der Request. Sie sollten die Bedeutung der einzelnen Felder aber dennoch verstehen können. Beantworten Sie dazu folgende Fragen:

Wie ist der Status des abgefragten Zertifikats?

good



Offensichtlich wurde das Zertifikat nicht revoziert und der Browser darf es verwenden. Im Folgenden wird auf einige sicherheitstechnische Aspekte von OCSP eingegangen.

Die OCSP-Response ist signiert. Wieso ist das wichtig? Anders gefragt: Welche Attacke wäre möglich, wenn die Response nicht signiert wäre?

Man könnte den Request abfangen und eine falsche Response dem Browser/Client schicken.

Die Seriennummer des abgefragten Zertifikats ist auch in der Response erhalten. Wieso ist das wichtig? Anders gefragt: Welche Attacke wäre möglich, wenn das nicht so wäre?

Damit die gleiche Response nicht immer wieder benutzt wird.

Die Response enthält auch zwei Timestamps (*thisUpdate* und *nextUpdate*). Diese definieren das Zeitfenster, während dem die Antwort gültig ist. Wieso ist das wichtig? Anders gefragt: Welche Attacke wäre möglich, wenn kein Zeitfenster enthalten wäre?

In diesem Zeitfenster könnte das Zertifikat ungültig werden.

Der Timestamp *thisUpdate* kann dabei ziemlich genau der aktuellen Zeit entsprechen oder auch ein älteres Datum aufweisen, je nachdem ob die OCSP-Response «frisch» erzeugt wurde oder eine früher gecachte Version verwendet wird.

Die obigen drei Fragen betreffen ganz typische Punkte von sicheren Protokollen und sind im Grundprinzip auch nicht OCSP-spezifisch: Man muss bei sicheren Protokollen solche Dinge immer beachten und geeignet lösen, damit es für einen Angreifer nicht möglich ist, Nachrichten zu manipulieren oder frühere Nachrichten wiederzuverwenden (Replay-Attacke).

Wie Sie vorhergehend gesehen haben, ist die OCSP-Response signiert – mit welchem Zertifikat? Dies sieht man ebenfalls anhand der OCSP-Response. Dort findet man Hinweise, dass das Subject des verwendeten Zertifikats *US / Let's Encrypt / R3* entspricht; dies ist das Intermediate-CA-Zertifikat, mit dem auch das Zertifikat des Webservers ausgestellt ist. Dies bedeutet, dass der Browser die Signatur der Response ebenfalls mit diesem Intermediate-CA-Zertifikat prüfen kann. Es gibt aber auch CAs, die verwenden ein separates Zertifikat für das Signieren der OCSP-Responses, wobei dann dieses Zertifikat in der Regel wiederum vom Intermediate-CA-Zertifikat ausgestellt ist, womit die Gültigkeit der Signatur dann ebenfalls inklusive der Certificate-Chain geprüft werden kann.

Betrachten wir noch den Fall, dass ein Angreifer den Private Key des Intermediate-CA-Zertifikats kompromittiert hat. Damit kann der Angreifer Zertifikate im Namen der CA ausstellen und sich als beliebiger Server ausgeben oder «perfekte» Man-in-the-Middle-Attacken auf Zertifikats-basierte Authentisierung (z.B. im Rahmen von TLS) durchführen.

Der Browser sollte also auch noch prüfen, ob das Intermediate-CA-Zertifikat noch gültig ist oder ob es von der CA revoziert wurde. Das kann man ebenfalls mit OCSP machen. Schauen Sie sich das Intermediate-CA-Zertifikat nochmals genauer an. Welche Prüfmethode für den Status des Zertifikats steht dort?

Dieses Mal CA Issuer

Authority Info (AIA)

Location	http://r3.o.lencr.org
Method	Online Certificate Status Protocol (OCSP)
Location	http://r3.i.lencr.org/
Method	CA Issuers

Die CA ermöglicht es also auch, den Status des Intermediate-CA-Zertifikats zu überprüfen. Analysieren Sie als nächstes nochmals die aufgezeichneten Nachrichten in *Wireshark*. Können Sie hier eine (wie auch immer geartete) Abfrage bezüglich des Status dieses Zertifikats finden?

Obwohl der Status des Intermediate-CA-Zertifikats also überprüft werden könnte, wird dies zumindest von Firefox nicht gemacht (die meisten anderen Browser verhalten sich ebenfalls so). Ohne weitere Massnahmen besteht somit im ganzen Zertifikatsprüfprozess eine Lücke, die von einem Angreifer ausgenutzt werden könnte. Man kann argumentieren, dass es aufgrund der bei einer CA hoffentlich vorhandenen Sicherheitsmassnahmen unwahrscheinlich ist, dass der Private Key des Intermediate CA-Zertifikats kompromittiert wird – vermutlich einiges unwahrscheinlicher als das Kompromittieren des Private Key irgendeines Webservers. Dennoch ist eine solche Attacke natürlich denkbar, und das Schadenspotential wäre sehr hoch.

Um dieses Problem zu entschärfen, bieten die Browser teilweise durch eigene Lösungen Abhilfe. Bei Firefox existiert beispielsweise ein sog. OneCRL-Mechanismus. Dies ist im Wesentlichen eine Certificate-Revocation-List, die revozierte Intermediate Zertifikate der CAs beinhaltet. Diese Liste wird regelmässig von Mozilla aktualisiert und von Firefox heruntergeladen, womit revozierte Intermediate-CA-Zertifikate im Browser erkannt werden können.

5.3 OCSP-Stapling

Wie im Unterricht besprochen implementieren die Browser bei OCSP einen Soft-Fail-Ansatz. Dies bedeutet, dass der Browser ein Zertifikat als gültig akzeptiert, wenn er keine OCSP-Response erhält. Ein wichtiger Grund für diesen Soft Fail Ansatz ist, dass sonst oft verwendete Einsatzszenarien nicht mehr funktionieren würden. In einem Hotel muss man sich z.B. oft zuerst mit einem erhaltenen Passwort bei einem Webportal anmelden, bevor man Zugang zum Internet erhält. Die Kommunikation mit dem Portal verwendet dabei üblicherweise HTTPS (und damit TLS). Um das Zertifikat des Portal-Servers zu prüfen, müsste der Browser nun den OCSP-Responder kontaktieren, der «irgendwo» im Internet steht, zu diesem Zeitpunkt aber noch nicht erreichbar ist, weil der Benutzer sein Passwort noch nicht eingegeben hat. Ein Hard-Fail-Ansatz würde ein solches Einsatzszenario verunmöglichen.

Der Soft-Fail-Ansatz bringt dafür Sicherheitsprobleme mit sich. Nehmen Sie nun an, der Angreifer hat das Zertifikat des Webservers *WS* kompromittiert, d.h. er kennt den zugehörigen Private Key. Wir nehmen zudem an, dass das Zertifikat bei der CA revoziert wurde, d.h. die CA wird bei einer OCSP-Response den Status *revoked* verwenden. Zudem hat der Webserver bereits ein neues Zertifikat erhalten. Der Angreifer führt nun eine Man-in-the-Middle-Attacke auf die TLS-Verbindung zwischen einem Browser und *WS* durch, indem er gegenüber dem Browser das alte, kompromittierte Zertifikat verwendet. Wie kann der Angreifer erreichen, dass das Zertifikat vom Browser akzeptiert wird?

Request wird vom Angreifer abgefangen und keine Response an den Client geschickt. Somit fallen wir in den Soft-Fail beim Client.

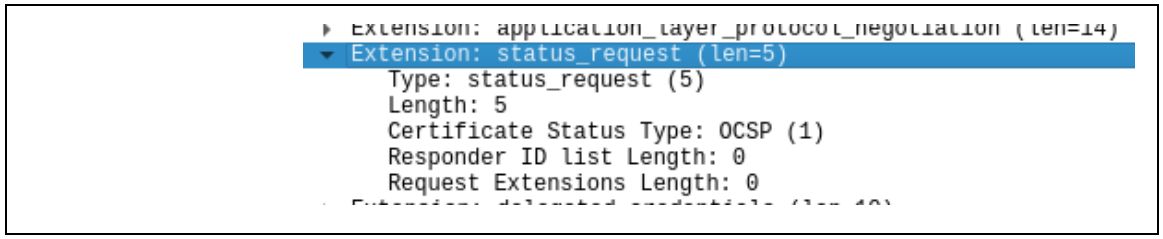
OCSP hat weitere Probleme. Eines ist die relativ hohe Last auf dem OCSP-Responder einer CA, da dieser im Prinzip jedes Mal, wenn eines der von der CA ausgestellten Zertifikate irgendwo verifiziert wird, kontaktiert wird. Und ein weiteres Problem von OCSP ist der Schutz der Privatsphäre: Da die CA bei jeder Prüfung eines Zertifikates vom Client kontaktiert wird, erfährt diese, welche Hosts (IP-Adresse des Clients) auf welche Server zugreifen.

Um mit diesen Problemen besser umzugehen, wurde OCSP-Stapling eingeführt. Die Idee ist, dass der Server selbst eine gültige OCSP-Response mitsendet, als Teil des TLS-Handshake. Dazu muss sich der Server selbst eine gültige OCSP-Response für sein eigenes Zertifikat beschaffen, unter «ganz normaler Verwendung» des OCSP-Protokolls und mittels Kommunikation mit dem OCSP-Responder. Diese OCSP-Response kann dann so lange verwendet werden, bis sie nicht mehr gültig ist (gemäss dem *nextUpdate* Eintrag, siehe oben). Dies reduziert die Last auf OCSP-Responder massiv und löst auch das Problem mit dem Schutz der Privatsphäre.

Um zu verstehen, wie OCSP-Stapling funktioniert, analysieren Sie im Folgenden einen TLS-Handshake mit der Website *https://www.grc.com*. Rufen Sie dazu die Website mit Firefox auf und zeichnen Sie den TLS-Handshake mit *Wireshark* auf. Beantworten Sie dann die folgenden Fragen.

Der Browser muss dem Server mitteilen, ob er OCSP-Stapling unterstützt. Dazu wird eine Extension in der *Client Hello*-Nachricht verwendet. Studieren Sie die verschiedenen Extensions in der Nachricht. Welche davon instruiert den Server, dass der Browser OCSP-Stapling unterstützt? Die Extension sollte selbsterklärend sein, weitere Informationen zu TLS-Extensions finden Sie im zugehörigen RFC[5].

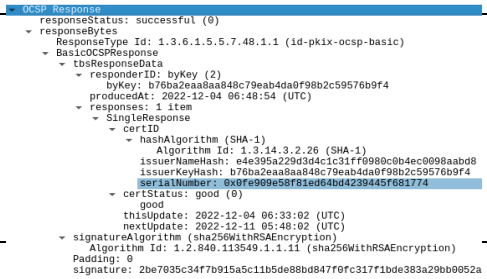
[5] https://tools.ietf.org/html/rfc6066



Unterstützt der Server OCSP-Stapling, so sendet er darauf die OCSP-Response als Teil des TLS-Handshake in einer spezifischen Handshake-Nachricht zum Browser. Wenn Sie die vom Server gesendeten Handshake-Nachrichten untersuchen, dann sollten Sie diese Nachricht einfach erkennen können. Wie lautet die Nachricht und was steckt in der Nachricht drin?

Handshake: Certificate Status

good



Die Nachricht enthält also die genau gleiche OCSP-Response, wie wenn der Browser den OCSP-Responder direkt kontaktiert hätte. Betrachten Sie als nächstes die Einträge für *thisUpdate* und *nextUpdate* und vergleichen Sie *thisUpdate* mit der aktuellen Zeit. Was schliessen Sie daraus? Hat der Server bei Ihrem Aufruf tatsächlich eine Anfrage an den OCSP-Responder gesendet?

Nein, gecached.

Der Server verwendet also eine frühere OCSP-Response. Dies zeigt, dass die Last auf den OCSP-Responder wirklich deutlich reduziert wird, da der Server typischerweise nur noch alle paar Tage eine OCSP-Anfrage sendet. Zudem erfährt die CA damit auch nicht, wie häufig der Server kontaktiert wird, was aus Gründen der Geheimhaltung von Geschäftsinformationen vorteilhaft ist.

Verifizieren Sie anhand der mit *Wireshark* aufgezeichneten weiteren Daten, ob Firefox auch noch selbst eine OCSP-Anfrage an den OCSP-Responder sendet. Finden Sie eine solche Anfrage? Und was schliessen Sie daraus?

Firefox hat keinen Request geschickt. Man kann davon ausgehen, dass der Intermediate noch gültig ist und nicht etwa bei Mozilla auf die revoked Liste oder sonst was steht.

Betrachten wir noch den Fall, dass der Server OCSP-Stapling nicht unterstützt oder keine aktuelle OCSP-Response besitzt. Letzteres kann auftreten, wenn der OCSP-Responder längerfristig nicht erreichbar ist oder auch wenn die Gültigkeit einer gecachten OCSP-Response gerade abgelaufen ist und der Server bei einer Kontaktaufnahme den OCSP-Responder zum Bezug einer aktuellen OCSP-Response nicht erreichen kann. In diesem Fall sendet der Server keine *Certificate Status* Nachricht als

Teil des TLS Handshake und der Browser kontaktiert den OCSP-Responder selbst, wie in Abschnitt 5.2 beschrieben. Und wenn das nicht klappt, dann gibt es wieder einen Soft Fail mit all den damit verbundenen Nachteilen.

Beurteilen Sie die Sicherheit von OCSP im Vergleich zu OCSP-Stapling bezüglich des Szenarios des kompromittierten Zertifikates, das bereits eingangs dieses Abschnitts verwendet wurde. Wir nehmen als wiederum an, dass der Angreifer das Zertifikat des Webservers *WS* kompromittiert, d.h. er kennt den zugehörigen Private Key. Wir nehmen zudem an, dass das Zertifikat bei der CA revoziert wurde, d.h. die CA wird bei einer OCSP-Response den Status *revoked* verwenden. Zudem hat der Webserver bereits ein neues Zertifikat erhalten. Erneut führt der Angreifer eine Man-in-the-Middle-Attacke auf die TLS-Verbindung zwischen einem Browser und *WS* durch, indem er gegenüber dem Browser das alte, kompromittierte Zertifikat verwendet. Wie kann der Angreifer erreichen, dass das Zertifikat vom Browser akzeptiert wird? Welchen Vorteil bietet hier OCSP-Stapling im Vergleich zu der direkten Verwendung von OCSP?

Netzauslastung ist geringer und schneller aufgrund vom cachen.

5.4 OCSP Stapling with Must-Staple

Einen Sicherheitsgewinn kann man nur mit einem Hard-Fail-Ansatz erreichen. Dazu dient eine zusätzliche *Must-Staple*-Option. Das resultierende *OCSP-Stapling with Must-Staple* funktioniert bzgl. Protokoll grundsätzlich genau gleich wir ohne Must-Staple, allerdings ohne den Fallback zum normalen Gebrauch von OCSP. Die OCSP-Response muss also zwingend als Teil des TLS-Handshake geliefert werden, ansonsten wird das Zertifikat nicht akzeptiert und die Verbindung nicht aufgebaut.

Als Beispiel verwenden wir *https://www.pragmatika.net*. Zeichnen Sie den TLS-Handshake wiederum mit *Wireshark* auf und verifizieren Sie, dass OCSP-Stapling grundsätzlich gleich verwendet wird wie im obigen Abschnitt.

Wie wird nun aber *Must-Staple* durchgesetzt? Dies steckt nicht im TLS-Handshake-Protokoll drin, sondern im Zertifikat des Servers. Betrachten Sie dazu die Extensions im Zertifikat des Servers. Dort finden Sie einen Eintrag *OCSP Stapling Required Yes*[6]. Dies ist nichts anderes als das Must-Staple-Flag, d.h. dieses weist den Browser an, dass die OCSP-Response zur Validierung dieses Zertifikats zwingend mit OCSP-Stapling erhalten werden muss und dass eine direkte Kontaktierung des OCSP-Responders nicht gemacht werden darf. Ein Soft Fail gibt es damit nicht mehr.

Betrachten Sie erneut die Man-in-the-Middle-Attacke von vorhin. Kann sie nach wie vor durchgeführt werden oder wird sie jetzt effektiv verhindert? Begründen Sie die Antwort!

Da eine Response verlangt wird, kann der Attacker nicht einfach keine Response schicken, was dazu führt, dass die Verbindung unterbrochen wird.

[6] Teilweise können Browser diesen Eintrag noch nicht sauber decodieren und melden etwas in der Art *Object Identifier (1 3 6 1 5 5 7 1 24)* mit dem Wert *30 03 02 01 05*. Dies ist aber nichts anderes als die «Rohform», wie das Must-Staple Flag in Zertifikaten eingetragen ist.

6 Cleanup

Wenn Sie das Image weiterverwenden möchten, so macht es Sinn, die *openssl*-Konfiguration wieder zurückzusetzen und den Virtual Host wieder zu deaktivieren:

```
$ /securitylab/certificates/unsetSL.sh
$ a2dissite lab_certificates
```

Praktikumspunkte

In diesem Praktikum können Sie **4 Praktikumspunkte** erreichen. Diese erhalten Sie, wenn Sie dem Betreuer Ihre Antworten auf die Fragen in der Praktikumsanleitung zeigen und diese Antworten mehrheitlich korrekt sind. Ebenfalls müssen Sie allfällige Kontrollfragen des Betreuers richtig beantworten können.