14.4 TLS/SSL

Schichteneinordnung

Das Transport Layer Security Protokoll (TLS) ist aus dem Secure Socket Layer Protokoll SSL [66] hervorgegangen. TLS ist ein Industriestandard für verschlüsselte Kommunikation. Es setzt auf der Transportebene auf. TLS ist ein Internet-Standard u.a. für sichere HTTP-Verbindungen, der von allen gängigen Web-Browsern unterstützt wird. Dabei wird das auf der Anwendungsebene angesiedelte HTTP Protokoll über Dienste, die von dem darunterliegenden TLS Protokoll angeboten werden, abgesichert. Dieses gesicherte Protokoll wird üblicherweise als HTTP über TLS, kurz HTTPS[13] bezeichnet. Hierbei wird, vereinfacht ausgedrückt, vom HTTP-Client ein sicherer TLS-Kanal zum Server aufgebaut, über den dann anschließend die HTTP-Daten transferiert werden. TLS ist jedoch nicht nur auf HTTP zugeschnitten, sondern bietet der Anwendungsebene (Schicht 7) Sicherheitsdienste an, die auf den Diensten der Transportebene aufsetzen, so dass über das TLS-Protokoll z.B. auch sichere FTP- oder E-Mail-Verbindungen aufgebaut werden können. Im Gegensatz zu SSL verwendet TLS das HMAC-Verfahren zur Berechnung der MAC-Werte. TLS verwendet ferner ein modifiziertes Schlüsselerzeugungsverfahren, wodurch eine stärkere Robustheit gegen Angriffe auf Hashwerte, die bei der Schlüsselgenerierung als Pseudozufallszahlengeneratoren eingesetzt werden, erreicht wird. TLS hat auch die Menge der Alert-Nachrichten erweitert, wobei alle Erweiterungen als fatale Warnungen eingestuft sind. Beispiele solcher Erweiterungen sind die Warnungen, dass eine unbekannte CA (Certification Authority) angegeben ist, dass die maximale Länge eines Record-Layer-Fragments überschritten worden ist, oder dass eine Entschlüsselungsoperation fehlgeschlagen ist.

Die aktuelle Version TLS 1.2 ist im RFC 5246 spezifiziert. Im März 2018 wurde der Internet-Draft von TLS 1.3 veröffentlicht[14]. Auf die Unterschiede zwischen TLS1.2 und TLS1.3 gehen wir am Ende der Ausführungen zu TLS noch kurz ein.

Implementierung

Implementierungen der Protokolle SSL und TLS stehen über Open Source Bibliotheken wie OpenSSL oder GnuTLS zur Verfügung. Des weiteren bieten viele Software-Plattformen, wie beispielsweise Windows in dem Secure Channel Package, Implementierungen beider Protokolle und SSL/TLS werden auch von den gängigen Browsern, wie Internet Explorer, Chrome, Firefox, Opera oder Safari unterstützt.

Abbildung 14.24 zeigt die Einordnung des TLS-Protokolls in die Schichtenarchitektur des TCP/IP-Modells.

[13] HTTP Secure.

[14] https://datatracker.ietf.org/doc/draft-ietf-tls-tls13/ gültig bis September 2018

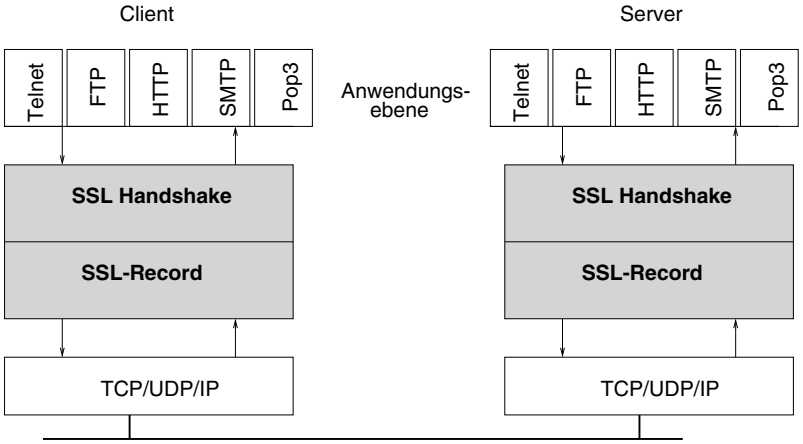

Abbildung 14.24: Schichteneinordnung des TLS-Protokolls

14.4.1 Überblick

Aufgaben

Die Hauptaufgaben von TLS sind die Authentifikation der Kommunikationspartner unter Verwendung von asymmetrischen Verschlüsselungsverfahren und Zertifikaten, die vertrauliche Ende-zu-Ende-Datenübertragung unter Nutzung eines gemeinsamen Sitzungsschlüssels und schließlich auch die Sicherstellung der Integrität der transportierten Nachrichten unter Nutzung von Message Authentication Codes. Wie schon IPSec so verwendet auch TLS für die bidirektionale Verbindung zwischen Client und Server zwei unterschiedliche Sitzungsschlüssel.

Verfahren

Die benötigten kryptografischen Verfahren und Hashfunktionen sind nicht a priori festgelegt, sondern werden pro Verbindung bzw. pro Sitzung, die auch mehrere Verbindungen umfassen kann, auf einfache Weise zwischen Client und Server abgesprochen. Anders als beim IPSec ist das Aushandeln der zu verwendenden Verfahren sowie der Austausch der benötigten Schlüssel ein integraler Protokollbestandteil. Wir werden darauf noch genauer eingehen. TLS legt ferner Rahmenvorgaben für die Verfahren zur Schlüsselerzeugung und der Übertragung authentifizierter Daten während der Phase des Verbindungsaufbaus fest. Beim Verbindungsaufbau sind Verfahren zur authentifizierten Verschlüsselung wie AES-GCM oder ChaCha20-Poly1305 zu verwenden und für den Bereich asymmetrischer Verfahren kann zwischen RSA und dem Diffie-Hellman-Verfahren gewählt werden.

Informationsaustausch

Obwohl in TLS eine Kombination aus asymmetrischen und symmetrischen Verschlüsselungstechniken eingesetzt wird, ist es kein Hybridverfahren zum Austausch eines symmetrischen Schlüssels unter Verwendung asymmetrischer Verfahren. Hierbei wird der geheime Schlüssel zwar geschützt, aber dennoch über ein unsicheres Medium übertragen. TLS versucht dagegen,

möglichst wenig geheime Information über das nicht vertrauenswürdige Transportsystem, das Internet, zu transportieren. Deshalb wird lediglich eine Basisinformation zwischen Client und Server vereinbart, mit der die beteiligten Partner dann dezentral ihre weiteren Geheimnisse, wie den gemeinsamen Verschlüsselungsschlüssel und die MAC-Schlüssel, berechnen können. Die beiden pro Sitzung verwendeten Sitzungsschlüssel sowie die beiden MAC-Schlüssel werden unter TLS also nie über das unsichere Netz übertragen.

Sitzung

Die in Abbildung 14.24 angegebene Einordnung von TLS verdeutlicht, dass TLS Aufgaben der Sitzungs- und Präsentationsschicht (Schichten 5 und 6) des ISO/OSI-Modells übernimmt. Ein wesentlicher Vorteil der Sitzungsschicht gegenüber der TCP-Ebene besteht darin, dass Zustandsinformationen über einen längeren Zeitraum und über verschiedene Einzelverbindungen hinweg gespeichert und für die Verwaltung genutzt werden können. Für das zustandslose HTTP-Protokoll, das für jeden Zugriff auf eine Webseite eine neue TCP-Verbindung aufbaut, bedeutet das, dass mehrere solcher Verbindungen zu einer Sitzung gebündelt und damit effizienter als die jeweiligen Einzelverbindungen verwaltet werden können.

TLS-Teilschichten

Wie Abbildung 14.24 zeigt, besteht das TLS-Protokoll im Wesentlichen aus zwei Teilen, nämlich dem Record- und dem Handshake-Protokoll. Zu den Aufgaben der Recordschicht gehört die Fragmentierung der Daten der Anwendungsebene in TLS-Records, deren Kompression, die Berechnung von MACs und der gemeinsamen Sitzungsschlüssel sowie die Verschlüsselung der TLS-Records. Auf der Handshake-Schicht findet die Authentifikation der Kommunikationspartner, das Aushandeln der zu verwendenden Verfahren und der Austausch benötigter geheimer Informationen statt.

ChangeCipher Spec

Neben diesen beiden Hauptprotokollen gehören zu TLS noch das Change Cipher Spec und das Alert Protokoll, die beide auch auf dem TLS-Record Protokoll aufsetzen. Das Change Cipher Spec Protokoll umfasst lediglich eine Nachricht bestehend aus einem Byte mit dem Wert 1. Mit dieser Nachricht werden die im Handshake ausgehandelten Verfahren als aktuell zu verwendende Verfahren übernommen.

Alert

Mit dem Alert-Protokoll können TLS-spezifische Warnungen an die Kommunikationspartner übermittelt werden. Eine Alert-Nachricht besteht aus zwei Bytes. Mit dem ersten Byte wird die Schwere der Warnmeldung angezeigt. Der Wert 1 beschreibt lediglich eine Warnung, während der Wert 2 einen fatalen Zustand signalisiert, was dazu führt, dass TLS die Verbindung sofort abbricht und auch keine neuen Verbindungen für diese Sitzung mehr eröffnet. Das zweite Byte beschreibt die Alarmsituation. Beispiele hierfür sind die Angabe, dass ein nicht korrekter MAC-Wert empfangen wurde, dass ein Zertifikat bereits abgelaufen oder zurückgerufen ist oder auch dass das

Handshake-Protokoll keine beidseitig akzeptierten Verschlüsselungsverfahren aushandeln konnte.

reservierte Ports

Um das TLS-Protokoll zwischen einem Client und einem Server abzuwickeln, müssen sich beide Partner zunächst über die Verwendung des Protokolls verständigen. Die erforderliche Information könnte zum Beispiel im Verlauf einer normalen TCP/IP-Sitzung übertragen und dabei der Einsatz von TLS vereinbart werden oder die Abstimmung über die Verwendung von TLS könnte ein integraler Bestandteil von Anwendungsprotokollen sein. Letzteres würde aber die explizite Änderung jedes Anwendungsprotokolls erfordern, was nicht praktikabel ist. In der Praxis wird ein sehr viel einfacherer Weg beschritten, indem für TLS-basierte Anwendungsdienste spezielle Portadressen reserviert sind, über die diese Anwendungsdienste abgewickelt werden. Tabelle 14.1 listet einige der wichtigsten, offiziell von der IANA reservierten Portadressen mit den daran gekoppelten Diensten auf.

Bezeichnung	Portadresse	Bedeutung
https	443	TLS-basiertes HTTP
ssmtp	465	TLS-basiertes SMTP
snntp	563	TLS-basiertes NNTP
telnets	992	TLS-basiertes Telnet
ftps	990	TLS-basierte FTP-Kontrollnachrichten
ftp-data	889	TLS-basierte FTP-Daten

Tabelle 14.1: Auswahl reservierter TLS-Portadressen

14.4.2 Handshake-Protokoll

Handshake

TLS ist ein zustandsbehaftetes Protokoll, so dass, wie bereits erwähnt, Sitzungen zwischen Kommunikationspartnern etabliert werden können. Ein Client kann zu einem Zeitpunkt mehrere solche Sitzungen zum gleichen oder zu verschiedenen Servern unterhalten. Es gehört zur Aufgabe der Handshake-Schicht, die Sitzungsinformationen von Client und Server zu koordinieren und konsistent zu halten.

Protokollschritte

In Tabelle 14.2 sind die Schritte des Handshake-Protokolls zusammengefasst. Im Verlauf dieses Protokolls werden die benötigten geheimen und

öffentlichen Informationen ausgetauscht, so dass damit die Kommunikationspartner in der Lage sind, die gemeinsamen MAC- und Sitzungs-Schlüssel dezentral zu berechnen.

Client		Server	Nachricht
1	$\longrightarrow$		ClientHello
2	$\longleftarrow$		ServerHello Certificate (optional) ServerKeyExchange (optional) CertificateRequest (optional) ServerHelloDone
3	$\longrightarrow$		Certificate (optional) ClientKeyExchange CertificateVerify (optional) ChangeCipherSpec Finished
4	$\longleftarrow$		ChangeCipherSpec Finished
5	$\longleftrightarrow$		Anwendungsdaten

Tabelle 14.2: TLS-Handshake-Protokoll

ClientHello

Um eine Verbindung zum Server aufzubauen, muss der Client zunächst seine ClientHello-Nachricht absenden. Mit dieser Klartextnachricht werden bereits Informationen ausgetauscht, die in späteren Schritten zur Berechnung der gemeinsamen Geheimnisse erforderlich sind. Die wichtigsten dieser Informationen umfassen eine Datenstruktur R_c bestehend aus einem 32-Bit Zeitstempel und einer 28-Byte Zufallszahl, einem Sitzungsidentifikator, und einer Prioritätenliste (Cipher Suite) mit denjenigen kryptografischen und Kompressionsverfahren, die der Clientrechner unterstützt. Eine solche Cipher Suite könnte beispielsweise wie folgt lauten: AES-128Bit, SHA2-MAC. Das bedeutet, dass zur symmetrischen Verschlüsselung der AES mit einer Schlüssellänge von 128 Bit und zur Berechnung der Hashwerte ein SHA2-MAC verwendet werden soll. Für TLS sind eine Reihe solcher Cipher-Suites vordefiniert, so dass nicht beliebige Kombinationen von Verfahren abgestimmt werden können. Der Sitzungsidentifikator besitzt genau dann einen Wert ungleich Null, wenn eine schon bestehende Sitzung genutzt oder eine frühere Sitzung wieder aufgenommen werden soll. Der Random-

R_c

wert R_c hat die Aufgabe einer Nonce und dient somit zur Abwehr von Wiedereinspielungsangriffen beim Schlüsselaustausch.

ServerHello

R_S

Der Server antwortet seinerseits mit einer Hello-Nachricht, die ebenfalls eine Datenstruktur R_s bestehend aus einem 32-Bit Zeitstempel und einer 28-Byte Zufallszahl sowie einer Liste von kryptografischen und Kompressions-Verfahren enthält. Diese Verfahren wählt der Server aus der Liste des Clients aus, vorausgesetzt, dass er sie selber ebenfalls implementiert hat. Damit sind die für die Sitzung zu verwendenden Verfahren ausgehandelt. Die von Client und Server jeweils unterstützten Verfahren werden in Konfigurationsdateien festgelegt, wobei die Reihenfolge der Listeneinträge deren Priorität bestimmt. Der Server sucht gemeinsame Verfahren mit möglichst hoher Priorität aus. Möchte der Client eine bestehende Sitzung nutzen, so sucht der Server den entsprechenden Sitzungsidentifikator in seinem Sitzungs-Cache. Falls er ihn findet und zusätzlich auch bereit ist, unter diesem Sitzungsidentifikator eine weitere Verbindung aufzubauen, so antwortet er mit dem gleichen Identifikator. Gibt der Server keine Angabe zum Sitzungsidentifikator zurück, zeigt er dadurch an, dass diese Sitzung von ihm nicht gespeichert wird und damit zu einem späteren Zeitpunkt vom Client auch nicht wiederaufgenommen werden kann.

Zertifikat

temporärer Schlüssel

Soll der Server authentifiziert werden, so muss er dem Client sein Zertifikat zukommen lassen, das dem in der Hello-Nachricht abgesprochenen Format zu entsprechen hat. In der Regel werden hierzu X.509 Zertifikate eingesetzt. Falls der Server auch eine Authentifikation des Clients fordert (CertificateRequest-Nachricht), antwortet der Client mit seinem eigenen Zertifikat in analoger Weise. In der optionalen CertificateRequest Nachricht gibt der Server X.500 Namen von denjenigen Certification Authorities (CAs) an, denen er vertraut, und teilt dem Client gleichzeitig mit, welche Verfahren er beherrscht (RSA, DSA etc.). Versendet der Server kein Zertifikat, so übermittelt er dem Client in der ServerKeyExchange-Nachricht einen temporären öffentlichen RSA-Schlüssel. Ein solcher Schlüssel wird zum Beispiel benutzt, wenn aufgrund von Exportrestriktionen der zertifizierte öffentliche Schlüssel des Servers wegen seiner zu großen Länge nicht verwendbar ist. In diesem Fall darf für den temporären RSA-Schlüssel auch nur ein 512-Bit Modul eingesetzt werden. Temporäre Schlüssel können mehrfach genutzt werden, sollten aber häufig (z.B. täglich oder nach einer festgelegten Anzahl von Transaktionen) geändert werden. Dies gilt natürlich insbesondere dann, wenn nur ein 512-Bit Modul verwendet wird. Anzumerken ist jedoch, dass in nahezu allen TLS Implementierungen heutzutage eine Server-Authentifizierung erfolgt. Der Server schließt die Übertragung der Nachrichtenfolge des Protokollschrittes 2 mit seiner ServerHelloDone-Nachricht ab.

Pre-Master Secret

Nach Erhalt der Daten aus Schritt 2 kontrolliert der Client zunächst die Gültigkeit des Server-Zertifikats. Weiterhin prüft er, ob die vom Server ausgewählte Liste von Verfahren tatsächlich mit seinen Angeboten verträglich ist. In diesem Fall übermittelt er mit der ClientKeyExchange-Nachricht dem Server eine geheime Basisinformation, das 48-Byte (384 Bits) Pre-Master Secret Pre. Haben sich die beiden Partner auf die Verwendung des RSA-Verfahrens verständigt, so verschlüsselt der Client das Geheimnis mit dem öffentlichen Schlüssel des Servers, $RSA(Pre, Public_S)$. Beim Einsatz des Diffie-Hellman-Verfahrens (DH) sendet der Client in dieser Nachricht nur seinen öffentlichen Schlüssel an den Server zurück. Man beachte, dass das DH-Verfahren in TLS nicht zur Berechnung des geheimen gemeinsamen Schlüssels, sondern zur dezentralen Berechnung des Pre-Master Secrets eingesetzt wird.

Master Secret

Das Pre-Master Secret Pre und die in den Hello-Nachrichten ausgetauschten Zufallszahlen R_c und R_s werden von Client und Server dazu verwendet, das 48-Byte Master Secret zu berechnen, aus dem dann die benötigten geheimen Schlüssel abgeleitet werden.

master_secret = PRF(Pre, master secret, R_c + R_s);

restliche Protokollschritte

Falls der Client ein Zertifikat vorweisen muss, sendet er die CertificateVerify-Nachricht, um dem Empfänger die Überprüfung des Zertifikats zu ermöglichen. Die Nachricht ist ein MAC über alle bislang im Protokoll ausgetauschten Nachrichten und wird mit dem privaten Schlüssel des Clients signiert.

Mit der ChangeCipherSpec-Nachricht zeigen sowohl der Client als auch der Server an, dass sie ab jetzt die ausgehandelten kryptografischen Verfahren verwenden. Über die Finished-Nachricht signalisieren beide die erfolgreiche Beendigung ihres jeweiligen Anteils am Protokoll. Diese Nachricht ist ebenfalls eine Datenstruktur bestehend aus einem MAC über alle bislang im Protokoll ausgetauschten Nachrichten. Falls die MACs beider Partner übereinstimmen, wird ohne weitere Acknowledgements die etablierte Verbindung akzeptiert und beide können jetzt mit der Übertragung von Anwendungsdaten beginnen.

14.4.3 Record-Protokoll

Record-Protokoll

Die Record-Schicht hat die Aufgabe, Datenpakete in TLS-Records mit einer maximalen Größe von 2^{14} Byte zu fragmentieren, optional zu komprimieren, diese Daten mit einem MAC zu hashen und sie zusammen mit ihrem MAC zu verschlüsseln. Abbildung 14.25 fasst die wesentlichen Aufgaben des Record-Protokolls zusammen. Die Kompression muss verlustfrei sein und darf die Länge des Fragments nicht um mehr als 1024 Bytes vergößern. Bei

sehr kleinen Blöcken kann es vorkommen, dass durch die Kompression der Block vergrößert anstatt verkleinert wird.

Nach der Verschlüsselung des MACs zusammen mit der komprimierten Nachricht darf die Länge des Kryptotextes die des Klartextes um höchstens 1024 Byte übersteigen. Der TLS-Record Header enthält unter anderem Informationen über die verwendete TLS-Version, den Content-Type, der angibt, welches TLS-Protokoll (Alert, Handshake, ChangeChipherSpec) zu verwenden ist, sowie die Länge in Bytes des Klartextfragments. Es ist übrigens wichtig und sinnvoll, dass eine Kompression vor der Verschlüsselung erfolgt, da ein Kompressionsverfahren im Prinzip darauf basiert, in dem Originaltext nach Mustern zu suchen und Duplikate zu eliminieren. Existieren nur wenige Muster, was ja bei einem verschlüsselten Text der Fall ist, versagt das Komprimieren und führt ggf. sogar zu einer Vergrößerung des Datenvolumens.

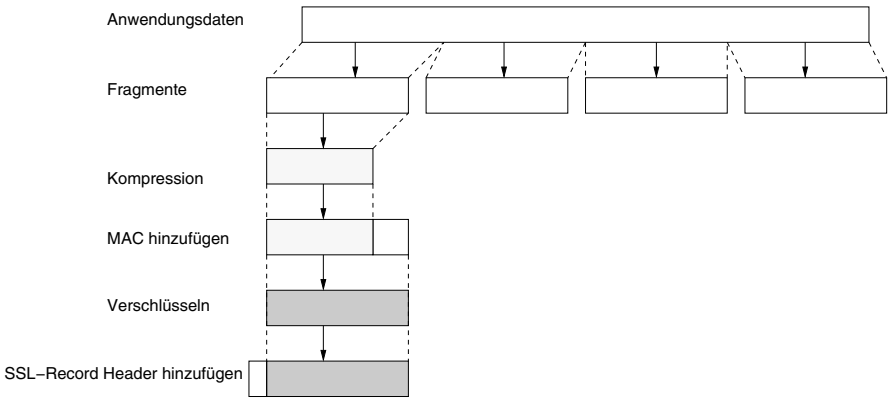

Abbildung 14.25: Aufgaben des TLS-Record Protokolls

Es bleibt somit zu klären, wie die benötigten Schlüssel, nämlich je ein MAC-Schlüssel für den Client und den Server sowie je ein geheimer Schlüssel für die Verbindung vom Client zum Server und umgekehrt, erzeugt und zwischen den Partnern abgestimmt werden.

Schlüsselerzeugung

Alle diese Schlüssel werden nach einem gleichartigen Schema sowohl vom Client als auch vom Server aus dem Master Secret (ms) abgeleitet. Dazu generiert das Protokoll so lange eine Folge von Schlüsselblöcken key_block, bis alle Schlüssel damit konstruiert sind.

Sitzungs- und Verbindungskonzept

Wie im Überblick bereits erwähnt und in den Protokollbeschreibungen gesehen, unterscheidet TLS zwischen Sitzungen und Verbindungen. Mit dem

bereits Erklärten können wir jetzt festhalten, durch welche Informationen eine Verbindung bzw. eine Sitzung unter TLS beschrieben ist.

Sitzungszustand

Eine Sitzung wird charakterisiert durch

- den Sitzungs-Identifikator: Das ist eine beliebige, durch den Server generierte Bytefolge, die eine aktive oder wieder aufnehmbare Sitzung charakterisiert.
- Zertifikat: Es handelt sich hierbei um ein X509.v3 Zertifikat des Partners.
- Kompressionsverfahren, wie z.B. zip,
- CipherSpec: Diese Spezifikation beschreibt die zu verwendenden Verschlüsselungs- sowie Hashverfahren und legt alle benötigten kryptografischen Attribute fest, wie zum Beispiel die Länge des Hashwertes.
- Master Secret, das zwischen Client und Server vereinbart ist.

Verbindungszustand

Eine Sitzung kann mehrere Verbindungen umfassen; jede Verbindung gehört zu genau einer Sitzung. Ein Verbindungszustand wird durch folgende Informationen beschrieben:

- Server- und Client-Randomzahlen der entsprechenden Verbindung,
- Server MAC-Schlüssel, der verwendet wird, wenn der Server Daten zum Client sendet,
- Client MAC-Schlüssel, der verwendet wird, wenn der Client Daten zum Server sendet,
- Serverschlüssel, der verwendet wird, wenn der Server Daten verschlüsselt zum Client sendet,
- Clientschlüssel, der verwendet wird, wenn der Client Daten verschlüsselt zum Server sendet,
- Initialisierungsvektor
- Sequenznummern: Jeder Partner verwaltet eigene Sequenznummern für gesendete und empfangene Datenpakete. Beim Senden bzw. Empfangen einer ChangeCipherSpec Nachricht werden die Sequenznummern der Verbindung auf 0 zurückgesetzt. Sequenznummern haben eine maximale Größe von $2^{64} - 1$.

An den Zuständen ist noch einmal abzulesen, dass die verwendeten Schlüssel nur jeweils für eine Verbindung gelten, während für alle Verbindungen einer Sitzung die gleichen Verfahren genutzt werden, so dass man bei der erneuten Verwendung einer bereits bestehenden Verbindung auf das Aushandeln der Verfahren im Handshake verzichten kann.

14.4.4 Sicherheit von TLS

TLS ermöglicht eine authentifizierte, vertrauliche und vor unbefugten Modifikationen geschützte Datenkommunikation zwischen zwei Kommunikationsendpunkten. Das Protokoll unterstützt verschiedene Authentifikationsschemata, die von einer wechselseitigen über eine nur einseitige Authentifikation bis hin zur vollständig anonymen Kommunikation reichen. Zu beachten ist, dass im letzten Fall auch die Vertraulichkeit der Kommunikation nicht notwendigerweise gewährleistet ist, da ggf. eine verschlüsselte Kommunikation zu einem Angreifer-Server aufgebaut wird. Die Daten sind dann zwar auf dem Transportweg verschlüsselt, so dass über einen passiven Angriff keine sensiblen Informationen erhältlich sind, aber die Verschlüsselung endet beim TLS-Endpunkt, also beim Angreifer-Server, bei dem die Daten dann im Klartext vorliegen. Diese Situation tritt beispielsweise ein, wenn bei einem Phishing-Angriff die Eingaben des Benutzers TLS-geschützt auf einen Angreifer-Server umgeleitet werden. Zu beachten ist ferner auch, dass die Art der Authentifikation allein vom Server bestimmt wird.

Authentifikation

Die Sicherheit des Handshake-Protokolls ist eine wesentliche Voraussetzung für die Sicherheit des gesamten TLS-Protokolls. Falls es nämlich einem Angreifer gelänge, dafür zu sorgen, dass die Kommunikationspartner schwache Verschlüsselungsverfahren oder schwache Schlüssel (z.B. 40 Bit) aushandeln, so könnte er anschließend mit großer Aussicht auf Erfolg versuchen, den verwendeten Kommunikationsschlüssel zu brechen. Für das Gelingen eines solchen Angriffs ist die Modifikation von Handshake-Nachrichten erforderlich. Das hätte aber zur Konsequenz, dass Client und Server in ihrer jeweiligen Finished-Nachricht unterschiedliche Hashwerte berechnen, da dieser MAC ja über alle von ihnen empfangenen und gesendeten Nachrichten gebildet wird. Die Finished-Meldungen würden deshalb wechselseitig nicht akzeptiert und der Verbindungsaufbau würde erfolglos terminieren. Der Angriffsversuch wäre somit abgewehrt.

Handshake-Protokoll

Um auch die Finished-Nachrichten gezielt zu modifizieren, benötigt der Angreifer das Master Secret, da dies zur Berechnung des MAC-Werts verwendet wird. Das Master Secret ist seinerseits aus dem Pre-Master Secret abgeleitet, so dass der Angreifer dieses Geheimnis kennen muss. Zu dessen Erlangung ist aber der private RSA- oder DH-Schlüssel des Servers notwendig. Bei geeignet großen Schlüsseln kann somit auch ein Angriff auf das Master Secret erheblich erschwert bzw. abgewehrt werden. Mit der Finished-Nachricht wird ein Hash-Wert über eine ganze Abfolge von Protokollschritten erstellt, so dass nicht nur die Integrität eines einzelnen Datenpakets, sondern vielmehr auch die der gesamten Nachrichtenabfolge sichergestellt ist. An dieser Stelle sei angemerkt, dass die eigentliche Authentifikation des Servers natürlich erst in diesem abschließenden Schritt

wirklich erfolgt ist. Denn mit dem korrekten Hash-Wert hat der Server die Kenntnis des korrekten geheimen Schlüssels nachgewiesen, der zu dem öffentlichen Schlüssel gehört, dessen Authentizität durch das Server-Zertifikat bestätigt wurde.

Perfect Forward Secrecy

Bereits in Kapitel 9.3 wurde auf die Bedeutung des Konzepts der Perfect Forward Secrecy (PFS) für die nachhaltige Sicherheit verschlüsselter Daten hingewiesen (vgl. Seite 418). Das TLS Protokoll ermöglicht die Umsetzung der Perfect Forward Secrecy, wenn im TLS-Handshake-Protokoll als Schlüsselaustauschverfahren das Diffie-Hellman-Verfahren verwendet wird. Da der für eine TLS-Verbindung benötigte Kommunikationsschlüssel auf der Basis temporär erzeugter DH-Schlüssel dezentral berechnet wird und der neue Kommunikationsschlüssel nicht von bereits etablierten Schlüsseln abhängig ist, wird gewährleistet, dass auch nach einer potentiellen Offenlegung des privaten Schlüssels, z.B. des Servers, daraus nicht nachträglich der Sitzungsschlüssel für verschlüsselte Kommunikation aus früheren TLS-Verbindungen, die aufgezeichnet wurden, entschlüsselt werden. Bis zum Sommer 2013 war jedoch die PFS Variante von TLS nur sehr wenig verbreitet, da diese Variante einen beträchtlichen zusätzlichen Overhead erfordert. Im Zuge der NSA-Affaire im Sommer 2013, bei der die massenhafte Aufzeichnung von unverschlüsselter, aber auch verschlüsselter Kommunikation, wie verschlüsselter TLS-Verbindungen, durch US amerikanische Geheimdienste auch einer breiten Öffentlichkeit bewusst wurde, gewann die Frage der Umsetzung des PFS-Konzepts bei TLS Verbindungen für Provider stark an Bedeutung. So fordert das BSI für die Bundesverwaltung als Mindeststandard die Verwendung von TLS 1.2 mit Perfect Forward Secrecy.

Klar ist auch, dass ein Man-in-the-Middle eine TLs-Verbindung übernehmen kann, wenn Client und Server sich nicht wechselseitig stark authentifizieren.

schwache Kryptografie

Die Sicherheit des Protokolls hängt zum einen von den verwendeten kryptografischen Verfahren ab, die die Kommunikationspartner im Handshake miteinander abstimmen, und zum anderen natürlich von der sicheren Implementierung der Funktionen, was im April 2014 durch den so geannten Heartbleed-Angriff auf OpenSSL-Software-Bibliotheken eindrucksvoll bewiesen wurde.

Heartbleed-Angriff auf OpenSSL

Sicherheitslücke

Anfang April 2014 wurde eine massive Sicherheitslücke in OpenSSL-Bibliotheken bekannt. Ursache für die aufgedeckte Schwachstelle ist ein Implementierungsfehler in der Heartbeat-Funktion von OpenSSL-Bibliotheken der Versionen 1.0.1 bis 1.0.1f. Die Schwachstelle besteht in einer fehlenden Überprüfung von Parametern. Die Verwundbarkeit wurde durch die so genannte Heartbleed-Attacke ausgenutzt. Da diese OpenSSL-Bibliotheken

sehr häufig als Basis von anderen Anwendungen genutzt werden, sind von der Schwachstelle neben Web-Servern wie Apache, auch Betriebssystemkomponenten oder Anwendungen wie IMAP, POP oder auch SMTP betroffen. Aufgrund der Vielzahl der angreifbaren Anwendungen verursacht die Beseitigung der Schwachstelle ganz erhebliche Aufwände, da private Schlüssel und Zertifikate erneuert, alte Zertifikate gesperrt, Passworte geändert und auf einer Vielzahl von Systemen Patches eingespielt werden müssen. Problematisch war zudem, dass die Sicherheitslücke bereits mit dem OpenSSL Release 1.0.1 vom 14.03.2012 verbreitet, aber erst mit Release 1.0.1.g am 07.04.2014 behoben wurde.

Heartbeat-Funktion

Ausgangspunkt für den Angriff ist die Heartbeat-Funktion in OpenSSL, die es ermöglicht, eine TLS Verbindung zwischen Client und Server aufrecht zu erhalten. Dazu sendet einer der Partner eine Nachricht mit beliebigem Inhalt an den jeweils anderen und der Empfänger sendet die Daten wieder zurück, um damit zu dokumentieren, dass die Verbindung noch in Ordnung ist. Um Daten des Anfragers (Client) in den Speicher des Empfängers (Servers) zu kopieren, wird das Kommando *memcpy(bp, pl, payload)* genutzt. Mit dem Kommando *memcpy* werden Daten kopiert, wobei der erste Parameter *bp* die Adresse angibt, wohin die Daten auf dem Server zu kopieren sind, der zweite Parameter *pl* spezifiziert die Adresse des zu kopierenden Bereichs und der dritte Parameter *payload* gibt die Länge der zu kopierenden Daten an. Abbildung 14.26 skizziert den Vorgang bei einem korrekten Heartbeat-Protokoll-Ablauf bezogen auf das Kopieren der Anfragedaten vom Client in den Pufferbereich des Servers. Der Server sendet die Daten in seinem Antwortpaket wieder an den Client zurück (vgl. Abbildung 14.26).

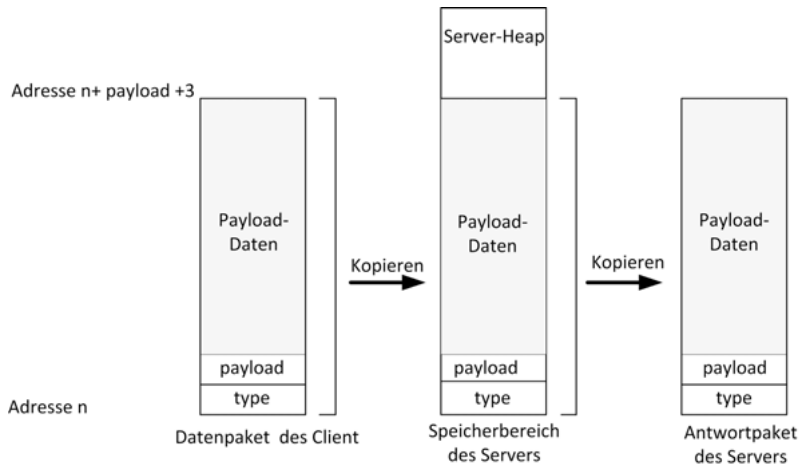

Abbildung 14.26: Korrekter Ablauf des Heartbeat-Protokolls

Die ausnutzbare Schwachstelle bestand darin, dass die Länge der übertragenen Daten nicht überprüft wird. Über den Parameter $payload$ gibt der Sender zwar an, wie lang das gesendete Payload ist, aber der Empfänger prüft nicht, ob die empfangenen Daten dieser Längenangabe gehorchen. Der Empfänger vertraut also darauf, dass der Sender eine korrekte Angabe sowohl in Bezug auf die Länge der Eingabe als auch auf die tatsächliche Eingabe macht.

Hier setzte der unten beschriebene Heartbleed-Angriff an. Anders als bei klassischen Buffer-Overflow-Angriffen wurde jedoch nicht eine zulange Eingabe an den Empfänger gesendet, sondern die tatsächliche Eingabe war deutlich kürzer als die angegebene Payload-Länge. Im RFC 6520 für die Heartbeat-Funktion wird angegeben, dass das Payload eine Länge von 16 KByte (also maximal 2^{14}) nicht überschreiten sollte. Da aber auch diese Angabe von vielen OpenSSL-Implementierungen nicht überprüft wird, kann auch ein 64 KByte großer Bereich als Payload angegeben werden, indem die maximale Payload-Größe, also $0xFFFF$, verwendet wird.

Angriff

Bei dem Heartbleed-Angriff[15] sendet der Angreifer ein Payload mit geringer Größe, z.B. 1 Byte, und behauptet gleichzeitig, dass er ein deutlich größeres Paket sendet, z.B. 16 KByte, d.h. der Parameter $payload$ wird entsprechend groß angegeben. Der Empfänger reserviert in seinem Speicher einen Pufferbereich P mit einer Länge wie in $payload$ gefordert. In dem Angriffsbeispiel reserviert der Empfänger also einen Bereich von 16 KByte. Er kopiert als nächstes die empfangenen Daten, im Beispiel ist das lediglich 1 Byte, in den reservierten Bereich P und sendet seine Antwort mit der Länge $payload$ zurück. Im Angriffsbeispiel wird somit eine Antwortnachricht von 16 KByte gesendet. Der Server versendet hierfür die Daten aus dem allokierten Speicherbereich P, also Daten, die der Server ggf. gerade bearbeitet. Aufgrund der speziellen Speicherverwaltung von OpenSSL stehen an der Speicherstelle nicht allgemeine Daten des Servers, sondern spezifische Daten der OpenSSL Implementierung, also beispielweise auch Zugangsdaten für Verbindungen anderer Nutzer, Passworte, Server- und Verbindungsschlüssel etc. Abbildung 14.27 fasst die wesentlichen Schritte dieses Angriffs noch einmal zusammen.

Fehlerbehebung

Der Implementierungsfehler betrifft die Funktion $tls1\_process\_heartbeat$ der Datei $t1\_lib.c$ der OpenSSL-Bibliothek. Behoben wurde der Fehler durch eine zusätzliche Kontrolle, bei der die Länge der Variable $payload$ mit der tatsächlichen Größe des empfangenen Payloads, das in der SSL3_RECORD-Struktur $(s3- > rrec)$ abgelegt ist, verglichen wird. Ist die Payload-Länge zu groß angegeben, so wird das Paket durch den Server verworfen.

[15] In der Schwachstellendatenbank CVE (Common Vulnerabilities and Exposures) unter http://cve.mitre.org/ wird die Schwachstelle unter dem Namen CVE-2014-0160 geführt.

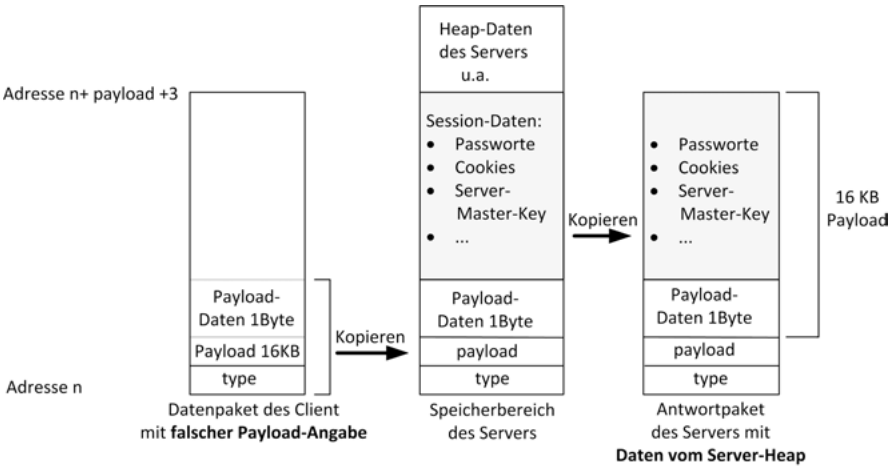

Abbildung 14.27: Ablauf eines Heartbleed-Angriffs

```
if (1 + 2 + payload + 16 > s->s3->rrec.length)
 return 0; /* silently discard per RFC 6520 sec. 4 */
```

Lessons learned

Durch den Heartbleed-Angriff wird, anders als bei den klassischen Buffer-Overflow-Angriffen, beim Server kein Speicherbereich unzulässig überschrieben. Der Angriff führt deshalb zu keiner Beeinflussung des Serververhaltens und bleibt für diesen verborgen. Der Angriff bzw. die Schwachstelle, die den Angriff ermöglichte, hat einmal mehr die Bedeutung der sicheren Programmierung mit u.a. Eingabeprüfungen verdeutlicht. Zudem hat der Angriff auch noch einmal den Mehrwert der Nutzung der Perfect Forward Secrecy (PFS) bei SSL/TLS verdeutlicht. Die Verwendung von PFS hätte die Schwachstelle zwar nicht geschlossen, aber das Schadensausmaß reduziert. Die Nutzung von PFS stellt sicher, dass ein mittels der Heartbleed-Attacke kompromittierter, aktueller privater Server-Schlüssel nicht genutzt werden kann, um damit dann auch verschlüsselte Verbindungen, die in der Vergangenheit aufgezeichnet wurden, im nachhinein zu entschlüsseln.

Grenzen

Auf TLs treffen alle Vorteile aber auch die Einschränkungen von Ende-zu-Ende-Sicherheitsprotokollen unterhalb der Anwendungsebene zu, die wir ja bereits angeführt haben. Da die Sicherheit des Protokolls in erster Linie von der sicheren Verwaltung der Schlüssel, insbesondere natürlich des privaten Schlüssels des Servers abhängt, kommt dessen sicherer Verwaltung eine besondere Bedeutung zu. Die meisten TLs-Server sind General Purpose Rechner, die nicht als vertrauenswürdig einzustufen sind. Da bei jedem Verbindungsaufbau der Private-Key des Servers verwendet werden muss,

ist klar, dass er im Hauptspeicher zu verwalten ist. Damit das sensitive Datum nicht im Zuge eines normalen Swap-Vorgangs im Zusammenhang mit der virtuellen Speicherverwaltung auf die Festplatte geschrieben wird, muss durch geeignete Lock-Konzepte ein solches Auslagern verhindert werden. Neben den allgemeinen Problemen weist das Protokoll aber auch noch einige protokollspezifische Problembereiche auf, die im Folgenden angerissen werden.

keine Verbindlichkeit

Da die übertragenen Daten nicht signiert werden, stellt TLS keine Verbindlichkeit von Aktionen sicher. Weiterhin werden auch keine Maßnahmen zur Abwehr von Verkehrsflussanalysen ergriffen, da nur die Nutzdaten in den TCP/IP-Paketen verschlüsselt werden.

Firewall

Das TLS-Protokoll kann nicht ohne weiteres zusammen mit Applikationsfilter-Firewalls betrieben werden. Ein dedizierter Proxy-Server eines Applikationsfilters wird von TLS als Mittelsmann eines Man-in-the-middle-Angriffs betrachtet, so dass der Verbindungsaufbau scheitern würde. Der TLS-Datenverkehr ist somit entweder zu tunneln oder es kann nur ein generischer Proxy-Firewall oder sogar nur ein Paketfilter verwendet werden, der die Nachrichten an die reservierten TLS-Ports ungehindert passieren lässt. Bei der Paketfilterlösung besteht natürlich die Gefahr, dass ein interner Angreifer diese reservierten Portadressen für andere Zwecke verwendet, ohne dass dies für den Paketfilter erkennbar ist. Ein TLs-Tunnel bedeutet, dass der Firewall den Datenverkehr nicht mehr analysieren und somit seinen Überwachungsfunktionen nicht mehr gerecht werden kann.

E2E Verschlüsselung

TLS etabliert eine verschlüsselte Kommunikation zwischen zwei Endpunkten (Ende-zu-Ende). Dies ist für viele heutige Szenarien noch angemessen. Mit der Zunahme komplexerer Transaktionen, an denen mehrere Server beteiligt sind, zum Beispiel im SOA oder Cloud-Kontext, wird zunehmend der Bedarf auftreten, einzelnen Servern nur Teile der Nachrichten sichtbar zu machen. Dafür sind erweiterte Konzepte erforderlich, beispielsweise analog zum Onion-Routing, damit nicht Sicherheitslücken dadurch entstehen, dass die Daten stets vollständig auf den jeweiligen Servern entschlüsselt werden, um weiterverarbeitet werden zu können.

Vertrauenswürdigkeit von SSL-Zertifikaten

Das Protokoll https, das zur sicheren Kommunikation zwischen zwei Endpunkten auf TLs aufsetzt, ist der De-Fakto-Standard für das sichere Browsen im Internet geworden. Mit dem TLS-Protokoll wird eine Vertrauensbeziehung zwischen den beteiligten Endpunkten aufgebaut, die auf den verwendeten SSL-Zertifikaten der Server und bei einer wechselseitigen Identifizierung auch der der Clients beruht. Die Vertrauenswürdigkeit der SSL-Zertifikate

und deren korrekte Validierung sind somit zentrale Eckpfeiler der Sicherheit in TLS-basierter Kommunikation. In heutigen Systemen wird diese Vertrauensbeziehung jedoch durch verschiedene Schwachpunkte in Frage gestellt. Im Folgenden werden wichtige Problembereiche deshalb abschließend diskutiert.

Eingespielte Zertifikate

Ein TLS-Nutzer muss den Informationen, die ihm zur Prüfung der Zertifikate vorliegen und im Browserfenster angezeigt werden, vertrauen können. Im Zusammenhang mit Problemen des Web-Spoofings (siehe Seite 160) haben wir jedoch bereits darauf hingewiesen, dass hier in heutigen Systemen eine Sicherheitslücke klafft, da es Angreifern möglich ist, dem Benutzer gefälschte Seiten und damit auch gefälschte Zertifikatsinformationen vorzugaukeln.

Vertrauenswürdige CAs

Das Vertrauenskonzept, das SSL-Zertifikaten zugrunde liegt, wird durch die gängige Praxis in heutigen Systemen stark ausgehöhlt, da über die bei der Installation vorgenommenen Voreinstellungen standardmäßig eine Vielzahl von Zertifizierungsinstanzen eingetragen sind, deren Zertifikaten automatisch vertraut wird, ohne dass der Benutzer selbst die Vertrauenswürdigkeit des Zertifikats überprüfen kann. Im etwas weiteren Sinn kann man diese Vorgehensweise wiederum als einen Verstoß gegen das Erlaubnis-Prinzip betrachten.

Gefälschte Zertifikate

Im Jahr 2011 gelang es Hackern, Zertifizierungsstellen, die SSL-Zertifikate ausstellen, zu kompromittieren, wodurch eine Vielzahl von gefälschten Zertifikaten für wichtige Domänen in Umlauf kam. Den Anfang machte ein Hacker-Angriff auf Registrierungsstellen des italienischen Zertifikat-Resellers Comodo. Dieser Angriff hatte zur Folge, dass der Angreifer sich über einen dieser kompromittierten Comodo-Registrare, den InstantSSL, gültige Zertifikate unter anderem für Microsoft, Mozilla, Yahoo, Skype und Google ausstellen lassen konnte. Diese Registrierungsstellen (Registrare) haben die Aufgabe, die Echtheit der Informationen im Zertifikatsantrag zu überprüfen, so dass sie nach erfolgreicher Prüfung von der Zertifizierungsstelle, in diesem Fall war das der Reseller Comodo, signiert werden.

DigiNotar

Noch gravierendere Ausmaße hatte der im Frühjahr des gleichen Jahres durchgeführte Angriff auf die niederländische Zertifizierungsinstanz DigiNotar[16]. Den Hackern gelang es, in die Infrastruktur von DigiNotar einzubrechen und alle acht Zertifizierungsstellen (CAs) unter ihre Kontrolle zu bringen. Die Angreifer ließen sich von den kompromittierten CAs über 500 gefälschte SSL-Zertifikate ausstellen. Zu den betroffenen Domänen, für die gefälschte Zertifikate ausgestellt wurden, gehören neben google.com, microsoft.com, facebook.com, twitter.com, aol.com, android.com und skype.com

[16] vgl. http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update/black-tulip-update.pdf.

auch die Domänen einiger Geheimdienste, wie www.sis.gov.uk (MI6) oder www.cia.gov. und www.mossad.gov.il. Der Einbruch wurde erst Monate später entdeckt, so dass erst dann die kompromittierten Zertifikate zurückgezogen werden konnten. Mit den gefälschten Zertifikaten konnten die Angreifer dafür sorgen, dass beispielsweise eine Vertrauensbeziehung zwischen einem Client und einem von ihnen kontrollierten Server aufgebaut und sensitive Information an den vermeintlich authentischen Server übermittelt wurde.

Zertifikatprüfung

Ein weiteres sehr gravierendes Problem betrifft die SSL-Zertifikatsprüfung auf den beteiligten Rechnern bei Anwendungen, die nicht Browserbasiert ausgeführt werden. Derartige Anwendungen treten heutzutage in vielfältigen Ausprägungen auf. So wird TLS beispielsweise genutzt, um mittels Fernzugriffen Cloud-basierte Infrastrukturen zu administrieren und Daten in Cloud-Storage Bereichen abzulegen. Weitere Anwendungsbereiche betreffen die TLS-basierte Übermittlung von Bezahlinformationen von einem E-Commerce-Server zu beispielsweise PayPal oder Amazon zur Bearbeitung der Bezahlungsinformation. Aber auch Verbindungen von Authentisierungsservern zu mobilen Anwendungen auf beispielsweise Android oder iOS Rechnern werden in der Regel TLS-gesichert aufgebaut. Diese Anwendungen implementieren normalerweise keinen eigenen TLS-Stack, sondern nutzen verfügbare TLS-Bibliotheken, wie OpenSSL, GnuTLS, JSSE,CryptoAPI oder auch Bibliotheken, die eine sichere Kommunikation auf einer höheren Netzwerkschicht anbieten und als eine Art Wrapper um die darunter liegenden TLS-Libraries fungieren. Beispiele sind cURL, Apache HttpClient oder auch urllib. Auch Web-Server Middleware, wie Apache Axis, Axis 2, oder Codehaus XFire bieten solche Wrapper auf einer höheren Abstraktionsschicht an

Sicherheitsanalyse

In [70] wurde ein breit angelegte Sicherheitsanalyse in Bezug auf die Validierung von SSL-Zertifikaten bei der Etablierung von authentisierten TLS-Verbindungen in nicht-Browserbasierter Software durchgeführt. Dabei wurden die gängigen Bibliotheken und Anwendungen für Linux, Windows, Android und iOS Systemen untersucht. Die Untersuchung hat ergeben, dass obwohl die genutzten TLS-Bibliotheken korrekt implementiert waren, ganz erhebliche Sicherheitsprobleme aufgetreten sind, da Software-Entwickler häufig die vielfältigen Optionen und Parameterkonstellationen dieser Bibliotheken missverständlich interpretieren. Als Beispiel nennen die Autoren die PHP Bibliothek des Flexiblen Payment Services von Amazon, der versucht, eine Verifikation des Hostnamen durchzuführen, indem der Parameter *CURLOPT_SSL_VERIFYHOST* der Bibliothek cURL auf *true* gesetzt wird. Der korrekte Parameterwert ist jedoch der Wert 2 und durch das Setzen des Wertes auf *true* wird der Parameter intern, ohne Fehlerhinweis nach außen, auf

Fehlerhafte Parameterwerte

1 gesetzt, wodurch keine Zertifikat-Validierung durchgeführt wird. Auch in der PHP Bibliothek des PayPal Payment-Standards wurde von den Forschern ein ähnlicher Bug entdeckt.
Als weiteres Beispiel für eine Fehlnutzung von Parametern in SSL-Libraries zeigen die Autoren Probleme mit der Bibliothek JSSE (Java Secure Socket Extension) auf, die unter anderem die API Funktion *SSLSocketFactory* zur Verfügung stellt. Diese API unterdrückt jedoch eine Zertifikatvalidierung, falls im Algorithmen-Feld des TLS-Client anstelle von der erwarteten Angabe https der Wert NULL oder ein leerer String steht. Zu den Software-Produkten, die eine solche Schwäche aufweisen, zählt der Apache HttpClient. Die Sicherheitsanalyse hat gezeigt, dass eine Vielzahl gängiger Software-Suiten, wie Apache Axis oder die EC2 Bibliothek von Amazon, Schwächen bei der Validierung von SSL-Zertifikaten aufweisen und damit anfällig gegen Man-in-the Middle Angriffe sind.

TLS1.2 versus TLS 1.3

TLS1.3 wurde im Vergleich zu TLS1.2 deutlich entschlackt und vereinfacht, so dass die Gefahr für Fehlkonfigurationen reduziert wurde. führt mit Verfahren wie Zero Round Trip Time (0-RTT) zu einer verbesserten Performanz, da in TLS1.3 nur noch ein Roundtrip an Nachrichten im Handshake benötigt werden, anstelle von zwei Roundtrips in TLS1.2. So können bei einer bekannten Seite direkt Daten zum Server übertragen werden (0-RTT). Der Client nutzt dafür einen bei der letzten Verbindung mit dem Server vereinbarten Schlüssel.

TLS1.3

Auch die Sicherheit wird unter TLS1.3. verbessert. Unterschiede zwischen den Versionen 1.2 und 1.3 sind u.a. dass in TLS1.3. keine proprietären symmetrischen Verschlüsselungsverfahren unterstützt werden. TLS1.3 hat die unsicheren Verfahren wie SHA-1, RC4, DES, 3DES, AES-CBC und MD5 entfernt. Alle verbleibenden Verfahren müssen zudem den AEAD Modus, siehe Seite 315, unterstützen. Damit wird der Design-Fehler aus TLS1.2, durch den erst der MAC über den Klartext berechnet und dann die damit erweiterte Nachricht verschlüsselt wird (MAC-then-Encrypt), behoben. Der Design-Fehler hat so genannte Padding-Oracle-Angriffe ermöglicht, bei denen ein Angreifer versucht, byteweise den Klartext zu erraten und die Integritätssicherung des Protokolls dem Angreifer verrät, ob er richtig geraten hat

In TLS1.3 ist ein Austausch des symmetrischen Schlüssels mittels RSA nicht mehr zulässig. Ein Schlüsselaustausch muss mit dem Diffie-Hellman-Vefahren erfolgen, dabei sollte DH mit elliptischen Kurven, also ECDH, genutzt werden. Alle verwendbaren Public-Key Verfahren unterstützen Perfect Forward Secrecy. Zudem wurden die Protokollnachrichten so geändert,

dass bereits nach der ersten Handshake-Nachricht, also nach dem Server-Hello, die übertragenen Nachrichten verschlüsselt werden. ECC-Verfahren gehören in der Version 1.3. zu den Basisverfahren und es wurden neue Signaturverfahren, wie ed25519, spezifiziert, DSA wurde entfernt.

14.5 DNSSEC

DNSsec

In Abschnitt 3.4.1 wurden die Sicherheitsprobleme des Domain Name Service (DNS) diskutiert. An Sicherheitserweiterungen von DNS wird bereits seit 1997 unter dem Namen DNSSEC (Domain Name System Security Extension) gearbeitet. Seit 2005 existiert mit den RFCs 4033, 4034 und dem Update von 2013 im RFC 6840 eine Spezifikation von DNSSEC.

Schutzziele

Schutzziele von DNSSEC sind die Authentizität, sowie die Integrität von DNS-Einträgen, um Angriffe wie DNS-Cache Poisoning abzuwehren. Demgegenüber sind die Gewährleistung der Vertraulichkeit von DNS-Daten, wie beispielsweise der Verschlüsselung von IP-Adressen, um die Privatsphäre zu schützen, oder die Abwehr von Denial-of-Service Angriffen keine Ziele von DNSSEC. DNSSEC nutzt asymmetrische Kryptografie zusammen mit kryptografischen Hashfunktionen (vgl. Kapitel 7), um die Schutzziele zu erreichen.

14.5.1 DNS-Schlüssel und -Schlüsselmanagement

Schlüssel

DNSSEC verwendet pro Domäne zwei Public-Key-Schlüsselpaare: den langlebigen Key-Signing-Key (KSK), bestehend aus einem privaten und einem öffentlichen Schlüssel, und den Zone-Signing-Key (ZSK), ebenfalls durch ein Schlüsselpaar festgelegt. Der private KSK sollte eine Länge von 2048 Bit besitzen; er ist 2 bis 4 Jahre lang gültig. Wie der Name bereits veranschaulicht, wird der private KSK dazu verwendet, andere Schlüssel zu signieren. Der private ZSK sollte eine Länge von mindestens 512 Bit besitzen; er ist lediglich 1 bis 2 Monate gültig. Mit seinem privaten ZSK signiert der Administrator einer Domäne alle DNS-Einträge in seiner Zonendatenbank. Die ZSK-Schlüssel können völlig autonom von den jeweiligen Administratoren einer Domäne verwaltet und erneuert werden. Eine Schlüsselerneuerung ist damit Domänen-lokal und einfach möglich. Dadurch kann man auch kürzere Schlüssel verwenden und damit die Effizienz der Validierungsoperationen deutlich gegenüber langen Schlüsseln verbessern. Da mit den ZSK-Schlüsseln alle DNS-Einträge signiert werden, die bei einem Lookup zu validieren sind, ist eine schnelle Validierung von großem Vorteil. Die KSK-Schlüssel werden demgegenüber domänenübergreifend verwendet, um Schlüsselmaterial zu signieren. Eine Schlüsselerneuerung erfordert eine domänenübergreifende Zusammenarbeit und ist damit aufwändig. Durch die

deutlich längeren KSK Schlüssel kann man die Erneuerungsintervalle ebenfalls deutlich verlängern, um diesem Aufwand Rechnung zu tragen. Da die KSK-Schlüssel lediglich zum Signieren anderer Schlüssel verwendet werden, ist der mit der Validierung verbundene Aufwand durch den langen Schlüssel tolerabel.

Signierte DNS-Einträge

Zur Veranschaulichung der Vorgehensweise betrachten wir folgendes Szenario: Gegeben sei die Domäne *.de* und deren Sub-Domäne *tum.de*. Im Folgenden erläutern wir zunächst, welche Schlüssel zum Signieren welcher Informationen verwendet werden, um die Integrität und Authentizität von DNS-Einträgen prüfbar zu machen. In einem zweiten Schritt erläutern wir dann, wie eine solche Prüfung abläuft und welche Informationen dafür erforderlich sind.
Wir gehen davon aus, dass für die beiden Domänen die beiden Schlüsselpaare ZSK_{de} und KSK_{de} bzw. $ZSK_{tum.de}$ und $KSK_{tum.de}$ bereits generiert worden sind. Wir beschreiben das generelle Schema aus Sicht der Domäne *tum.de*.

- Jeder DNS-Eintrag in der Zonendatenbank von *tum.de* wird vom Administrator der Domäne mit dem privaten Zonenschlüssel $ZSK_{tum.de}$ signiert. Zur Ablage dieser Information wird in DNSSEC ein neuer Datentyp eingeführt, das sind die RRSIG-Einträge.
- Den zugehörigen öffentlichen $ZSK_{tum.de}$ Schlüssel, der für die Verifikation der signierten DNS-Datenbankeinträge der Domäne *tum.de* benötigt wird, signiert der Administrator von *tum.de* mit dem privaten $KSK_{tum.de}$ Schlüssel seiner eigenen Domäne. Die Zonenschlüssel werden also selbst signiert.
- Der Administrator stellt die beiden öffentlichen Schlüssel, also die öffentlichen Schlüssel von $ZSK_{tum.de}$ und $KSK_{tum.de}$, die zur Verifikation der Signaturen erforderlich sind, in speziellen DNS-Datenbank-Einträgen zur Verfügung. Dazu führt DNSSEC den Datentyp DNSKEY ein. D.h. die Zonendatenbank enthält zwei spezielle DNSKEY Einträge für die beiden öffentlichen Schlüssel. Zusätzlich wird die Signatur des selbst signierten, öffentlichen Schlüssels $ZSK_{tum.de}$ in einem RRSIG-Eintrag zur Verfügung gestellt.
- Damit bei einer DNS-Anfrage die so signierten DNS-Einträge von Dritten verifiziert werden können, wird ein Nachweis über die Vertrauenswürdigkeit des öffentlichen Schlüssels $KSK_{tum.de}$ benötigt, mit dem die Schlüssel-Signaturen verifiziert werden können. Hierzu kommt nun die hierarchische Struktur des DNS ins Spiel. In unserem Szenario ist *.de* die übergeordnete Domäne von *tum.de*. Der Administrator von *.de* signiert mit seinem privaten Zonen-Schlüssel ZSK_{de} den Hashwert