

Kapitel 12: Netzsicherheit - Schicht 4: Transport Layer SSL / TLS

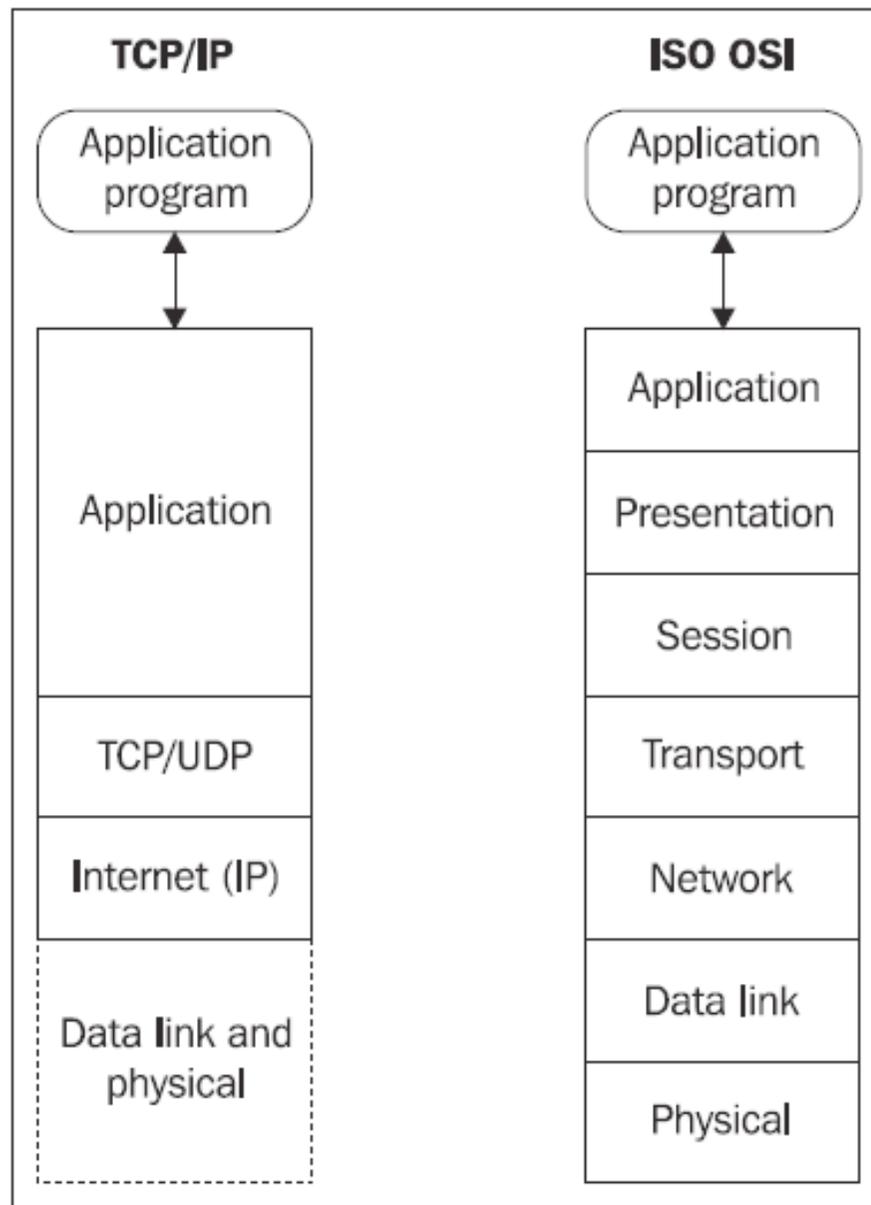


- Transport Layer: Funktionen
- Secure Socket Layer (SSL) & Transport Layer Security (TLS) - Historie
- SSL / TLS Protokoll-Architektur
 - SSL / TLS Record Protocol
 - SSL / TLS Handshake Protocol
 - Schlüsselerzeugung
- SSL / TLS Anwendung
- SSL / TLS Schwächen bzw. Vulnerabilities

- Nach OSI: Transportdienst zwischen Endsystemen; Ende-zu-Ende
 - Zuverlässiger Transport von
 - Nachrichten der Endsysteme

- In der Internet-Welt: Ende-zu-Ende-Verbindung zwischen Anwendungen
 - OSI-Schichten 5, 6 und 7 fallen in der Anwendungsschicht zusammen
 - Ports definieren die Prozesse (Dienste) der Anwendungsschicht

- Sicherungsprotokolle der Transportschicht
 - setzen auf TCP oder UDP auf
 - realisieren zum Teil die Funktionalität der Sitzungsschicht
 - liegen zwischen Transport Layer und Application Layer



Bildquelle: codeguru.com

- Ab 1994 ursprünglich entwickelt, um HTTP-Verkehr zu sichern (https); entwickelt von Netscape und ab SSL v2 in deren Browser integriert
- 1995 Internet Explorer mit PCT (Private Communication Technology)
- SSL v3: Protokollverbesserungen (aus PCT) und de-facto Standard
- Kann beliebige Anwendungen sichern (nicht nur HTTP)
- IETF entwickelt basierend auf SSL seit 1996 Transport Layer Security (TLS)
 - SSL gehört der Firma Netscape
 - TLS ist eine IETF-basierte, freie Spezifikation
 - TLS 1.0 und SSL 3.0 sind nahezu identisch
 - SSL und TLS werden häufig synonym gebraucht
 - Aktuell: TLS v1.2 (seit 2008)

SSL/TLS Einordnung

Anwendungsschicht				Anwendung
SSL Application Data Protocol	SSL Alert Protocol	SSL Change Cipher Spec Protocol	SSL Handshake Protocol	SSL
SSL Record Protocol				
Transportschicht				Netz
Netzwerkschicht				
Verbindungsschicht				

■ Authentisierung

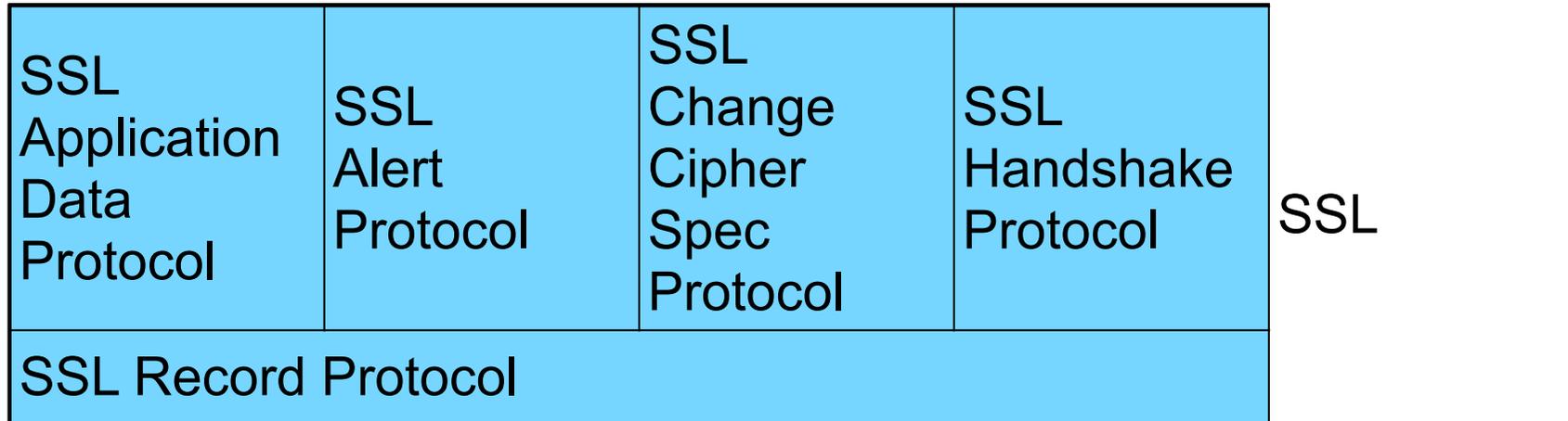
- Vor der eigentlichen Kommunikation ist eine Authentisierung möglich
- Einseitig oder auch zweiseitig:
 - Nur Client prüft Server (z.B. HTTPS bei Online-Banking)
 - Nur Server prüft Client (eher unüblich)
 - Client und Server prüfen sich gegenseitig (z.B. Intranet-Zugang mit Client-Zertifikat)

■ Vertraulichkeit der Nutzdaten

- Nur, falls während des Sitzungsaufbaus vereinbart
- Verschiedene (symmetrische!) Verschlüsselungsverfahren: RC2, RC4, DES, 3DES, DES40, IDEA, AES

■ Integrität der Nutzdaten

- Kryptographischer Hash-Wert, parametrisiert mit Schlüssel: HMAC
- Algorithmen: MD5, SHA



■ Application Data Protocol

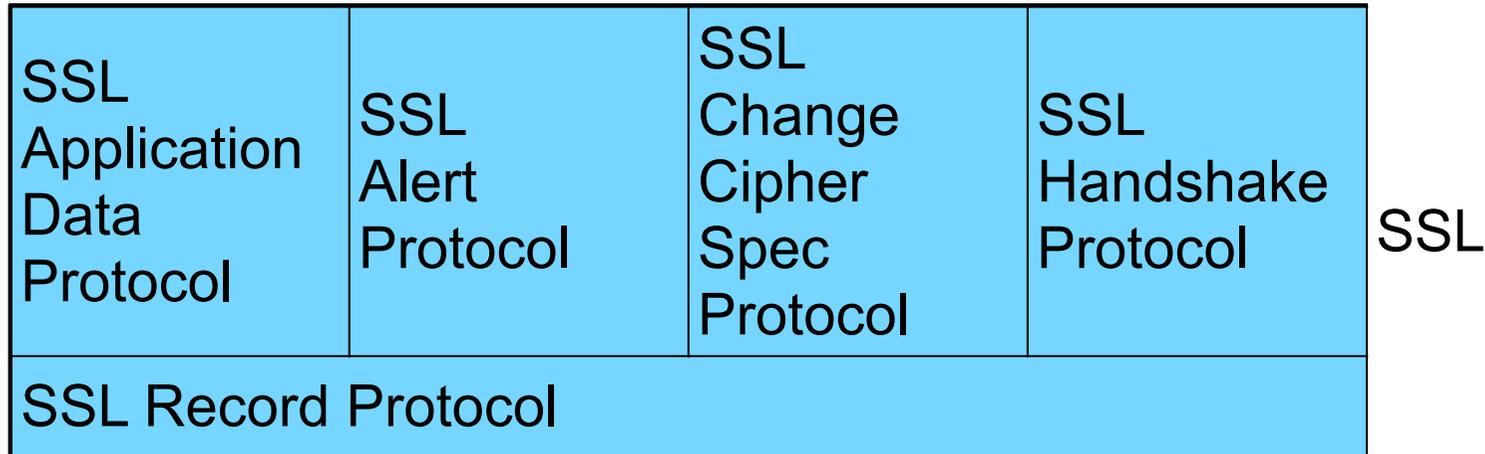
- Datenübermittlung zwischen Anwendung und SSL
- Zugriff auf Record Protocol

■ Alert Protocol:

- Warn- und Fehlermeldungen

■ Change Cipher Spec Protocol

- Änderung der Krypto-Verfahren
- Initialisierung und Einigung auf neu zu verwendende Verfahren



■ Handshake Protocol:

- Authentisierung
- Schlüsselaustausch
- Vereinbarung der Parameter

■ Record Protocol

- Fragmentierung
- Kompression der Klartext-Daten (optional)
- Verschlüsselung (optional)
- Integritätssicherung (optional)

1. Fragmentierung der Nutz-Daten in max. 214 Byte
2. Kompression der Daten (Default-Algorithmus null)
3. Integritätssicherung mittels MAC
4. Verschlüsselung

7	15	23	31
Type	Major Version	Minor Version	Length
Length	Data		

■ Type

- Change Cipher Spec (20)
- Alert (21)
- Handshake (22)
- Application Data (23)

■ Major und Minor Version (z.B. 3, 2 für TLS 1.1)

■ Length: Länge der Daten in Byte

SSL/TLS Record Protocol

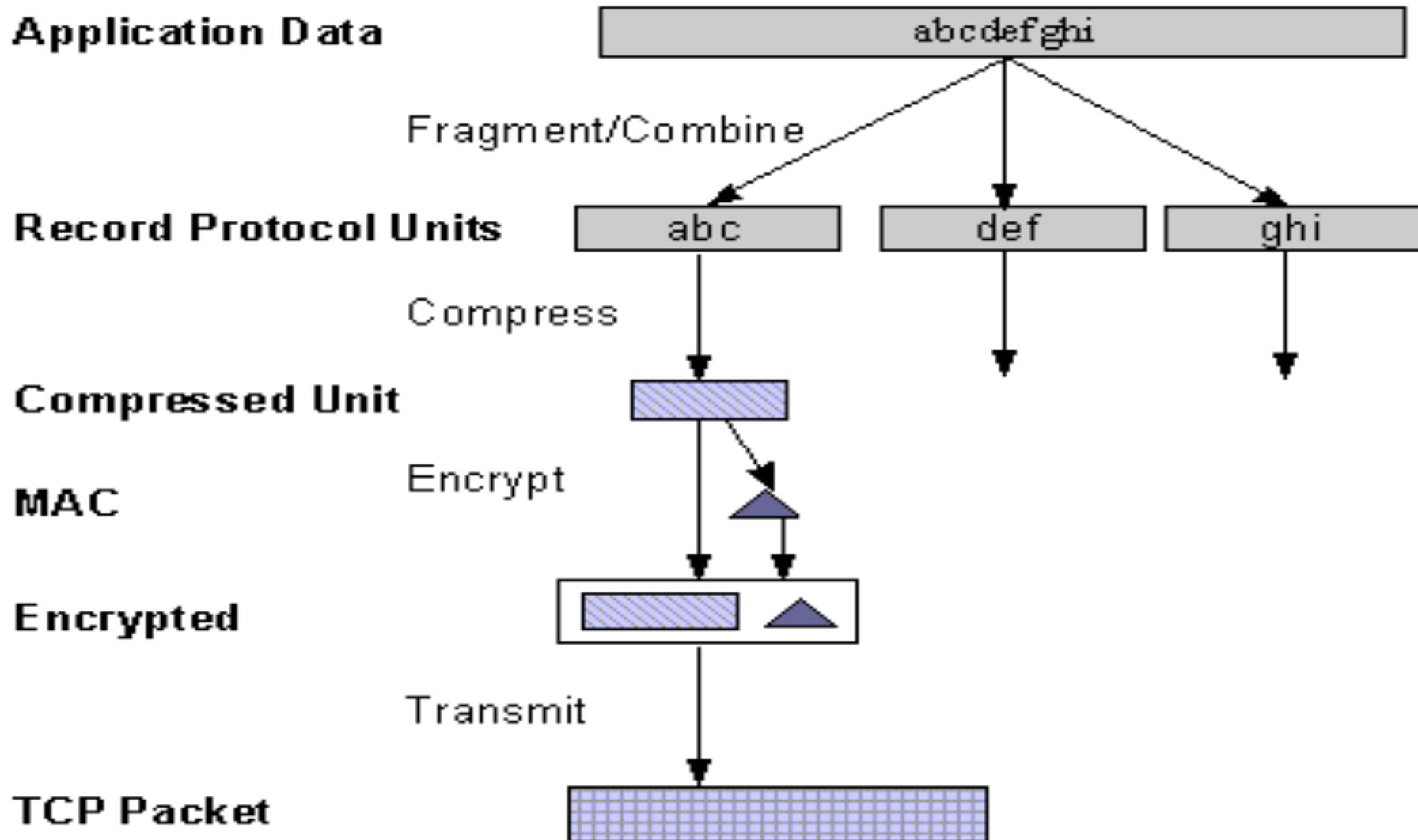
7	15	23	31
Type	Major Version	Minor Version	Length
Length	Data		

■ Sender

1. Fragmentierung der Nutzdaten in max. 214 Bytes (16 kB)
2. Kompression der Daten (Default-Algorithmus null)
3. Integritätssicherung mittels HMAC
4. Verschlüsselung

■ Empfänger:

- Entschlüsselung; Integritäts-Check; Dekompression; Defragmentierung; Auslieferung an höhere Schicht



Bildquelle: Ralf S. Engelschall, Apache mod_ssl Dokumentation

- Zweck: Authentisierung, Algorithmenauswahl, Schlüsselmaterial

Alice Client

Bob Server

ClientHello

enthält Liste der vom Client unterstützten Algorithmen

- Server wählt in Hello Nachricht Algorithmen
- Zertifikat zur Authentisierung
- Schlüsselmaterial (PreMaster Secret)
- Anforderung an den Client zur Authentisierung mittels Zertifikat

- ServerHello
- [ServerCertificate]
- [ServerKeyExchange]
- [CertificateRequest]
- ServerHelloDone

- [ClientCertificate]
- ClientKeyExchange
- [CertificateVerify]
-
- [ChangeCipherSpec]
- Finished

- Schlüsselmaterial (PreMaster Secret)

- [ChangeCipherSpec]
- Finished

SSL/TLS Handshake Protocol: Schlüsselgenerierung

- Schlüssel werden aus dem PreMasterSecret abgeleitet
- PreMasterSecret vom Client erzeugt; für Server verschlüsselt in ClientKeyExchange übertragen

- PreMasterSecret (variable Länge) wird erzeugt:
 - RSA: Zufallszahl; mit dem öffentlichen Schlüssel des Servers verschlüsselt vom Client übertragen
 - Diffie-Hellman: Übertragung der Diffie-Hellman Gruppe unverschlüsselt; falls nicht schon in Zertifikat enthalten; Erzeugung des PreMasterSecret über Diffie-Hellman Verfahren

- MasterSecret (immer 48 Byte) wird aus PremasterSec. erzeugt
 - $\text{MasterSecret} = \text{PRF}(\text{PreMasterSecret}, \text{„Master Secret“}, \text{ClientHello.random} + \text{ServerHello.random})$

- KeyBlock = PRF (SecurityParameter.MasterSecret, „key expansion“, SecurityParameter.ServerRandom + SecurityParameter.ClientRandom)
- Der KeyBlock wird in folgende Teilblöcke zerlegt
 - client_write_MAC_secret [SecurityParameter.HashSize]
 - server_write_MAC_secret [SecurityParameter.HashSize]
 - client_write_key [SecurityParameter.KeyMaterialLength]
 - server_write_key [SecurityParameter.KeyMaterialLength]
- SSL erlaubt Schlüsselerzeugung auch ohne Authentisierung
 - In diesem Fall Man-in-the-Middle-Attack möglich und nicht erkennbar

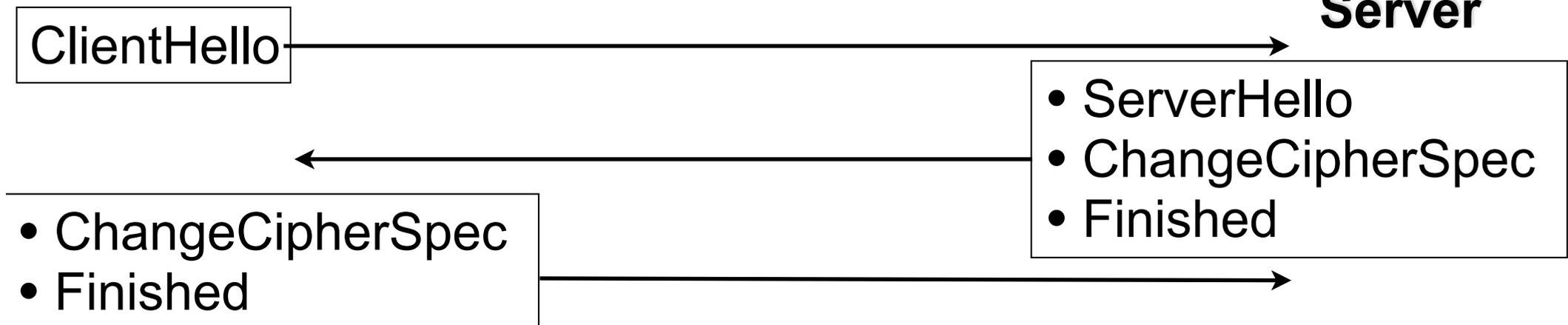
- Pseudo-Random Function (PRF); gebildet aus MD5 und SHA
- PRF soll Sicherheit bieten auch wenn MD5 oder SHA „gebrochen“ werden
- Expansionsfunktion $P_hash(secret, seed)$
 - Durch iterative Anwendung Schlüsselmaterial in beliebiger Länge
 - $P_hash(secret, seed) = \text{HMAC_hash}(secret, A(1) | seed) |$
 $\text{HMAC_hash}(secret, A(2) | seed) |$
 \dots
 $\text{HMAC_hash}(secret, A(n) | seed)$ mit
 - $A(0) = seed$
 $A(i) = \text{HMAC_hash}(secret, A(i-1));$
- $\text{PRF}(secret, label, seed) = P_MD5(S1, label + seed) \text{ XOR } P_SHA-1(S2, label + seed)$
 - mit secret zerleat in zwei Teilstrings S1 und S2

- TLS v 1.2 spezifiziert in RFC 5246 (2008); Änderungen:
 - Pseudo-Random Function:
SHA-256 löst Kombination aus MD5 und SHA ab
 - AES eingeführt
 - Erweiterungen bei der Spezifikation von Signatur und Hash-Funktionen durch Client und Server
 - Server-Name-Indicator (kommt später)

- Erlaubt Wiederverwendung und Duplizierung eines bestehenden Sicherheitskontextes
- HTTP 1.0; Jedes Item einer Webseite wird über eigene TCP Verbindung übertragen

Alice Client

Bob Server



- ClientHello enthält SessionID der zu duplizierenden Session
- Falls Server SessionID bei sich findet und mit Duplizieren einverstanden ist, sendet er SessionID in ServerHello zurück

- Rund 25 verschiedene Mitteilungen, z.B.
 - Beendigung der Session
 - Fehler in der Protokollsyntax (decryption failed, etc.)
 - Probleme mit der Gültigkeit von Zertifikaten

- Unterscheidet zwischen
 - Warnungen
 - Fehlern

- Fehler führen zu sofortigem Verbindungsende

■ Auswahl an SSL gesicherten Diensten

Port	gesicherter Dienst	Protokoll
443	HTTP	https
465, 587	SMTP (Mail)	ssmtp oder smtps
585,993	IMAP	imap4-ssl
636	LDAP	ldaps
989, 990	FTP	ftps
992	Telnet	telnets
995	POP3	pop3s

- HTTPS ist aufgrund der weiten Verbreitung von HTTP das häufigste SSL-Einsatzgebiet
- Problem bei virtuellen Webservern (z.B. Apache VHost):
 - Pro IP-Adresse nur 1 Zertifikat möglich
 - Zum Zeitpunkt des SSL-Handshakes liegen noch keine im HTTP-Request enthaltenen Angaben zum gewünschten VHost vor
- Gelöst mit TLS v1.2: Server Name Indication übermittelt gewünschten Servernamen bereits beim Verbindungsaufbau
- Aber: Verzögerung zwischen Spezifikation (RFC 5246), Implementierung (z.B. OpenSSL) und Rollout (z.B. Integration in Linux-Distributionen)

■ Performance

- Verbindungsaufbau ist rechenintensiv und damit langsamer
- Hohe Belastung für Server mit vielen Clients
- (Symmetrische) Verschlüsselung benötigt nur wenig Rechenzeit
 - Aber oft Verzicht auf Kompression (Entropiereduzierung!)

■ Unter Umständen keine Ende-zu-Ende-Verschlüsselung

- Bei Kommunikation über mehr als zwei Stationen erhält jede Zwischenstation den Klartext

■ Usability in der Praxis suboptimal

- Beispiel HTTPS-Zertifikatsprüfung in aktuellen Browsern: Häufig eher als lästig statt als hilfreich empfunden

■ Immer wieder Implementierungsfehler in SSL-APIs

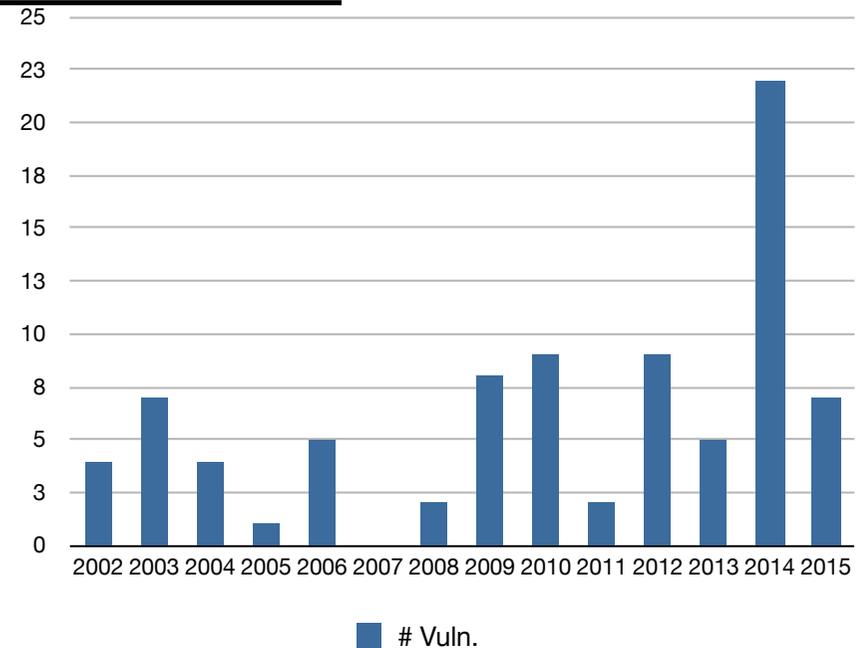
- Liste analysierter / behobener Sicherheitslücken:
<http://www.openssl.org/news/vulnerabilities.html>

- Vulnerabilities pro Jahr:
2014: 22 !!!
2015: bisher bereits 7

- Zum Teil gravierende Auswirkungen:

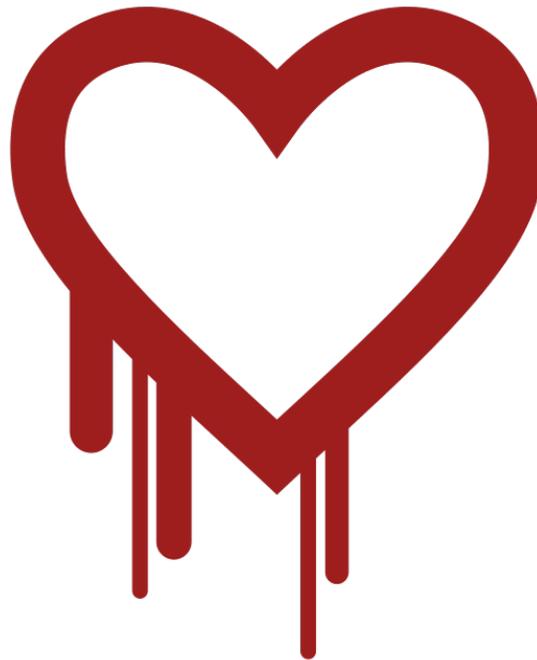
- Buffer Overflows
- Ungültige Zertifikate werden für gültig gehalten
- Crashes
- Memory Leaks
- Plaintext Recovery
- DoS

- Aber: Updates/Patches i.d.R. sehr zeitnah verfügbar



- Überwiegend implementierungsspezifische Angriffe
- Andere Varianten nur bedingt erfolgsversprechend:
 - Brute Force: Gesamten Schlüsselraum durchsuchen
 - Known Plaintext Attack: Viele Nachrichtenteile sind vorhersagbar, z.B. HTTP GET-Befehle (hat aber nur bei schwacher Verschlüsselung Aussicht auf Erfolg)
- Fehler im Protokoll-Konzept
 - Z.B. TLS & SSLv3 renegotiation vulnerability (November 2009)

■ SSL Heartbleed Logo



[„Heartbleed“ von Leena Snidate / Codenomicon - <http://heartbleed.com/heartbleed.svg>. Lizenziert unter CC0 über Wikimedia Commons - <https://commons.wikimedia.org/wiki/File:Heartbleed.svg#mediaviewer/File:Heartbleed.svg>]

- Sicherheitshinweis am 7.4.14 vom OpenSSL-Team
- SSL-Heartbeat Protokollerweiterung
 - Überprüfung ob Verbindung zum Server noch besteht
 - Client kann bis zu 16 kB beliebiger Payload schicken
 - Server antwortet mit Payload
- Implementierungsfehler
 - Angegebene Länge der Daten wird nicht mit tatsächlich übertragener Payload verglichen
 - Es werden Daten der angegebenen Länge zurückgeschickt (über den Eingabepuffer hinaus)
 - Buffer Overread
- Code wurde im Dezember 2011 in GIT-Repository gestellt
- Erstmals am 14. März 2012 in Version 1.0.1 veröffentlicht
- Schwachstelle bestand für 27 Monate!!!!



- Arbeitsspeicher des Servers kann ausgelesen werden
 - Schlüssel
 - Zertifikate und zugehörige private Schlüssel
 - Benutzernamen und Passwörter
 -
- Hinterlässt keine Spuren in den Logs
- Damit „beliebig“ viele Versuche möglich
- Folgen:
 - OpenSSL Aktualisieren
 - Serverzertifikate, Schlüssel und ggf. Passworte erneuern!!!!

- *Catastrophic is the right word. On the scale of 1 to 10, this is an 11.*
Bruce Schneier

- Schwachstelle am 14.10.14 veröffentlicht
- Padding Oracle On Downgraded Legacy Encryption (POODLE)
- Zweistufiger Man in the Middle Angriff
 - Angreifer erzwingt Fallback auf SSL 3.0 auf Client und Server-Seite
 - Schleust Code beim Client ein und greift dann die Verschlüsselung im Cipher Block Chaining Mode an
- SSL 3.0:
 - Zuerst wird MAC berechnet; MAC an M angehängt, dann Padding
 - Danach wird verschlüsselt
 - MAC then Encrypt Design (hat sich als problematisch erwiesen)

■ SSL 3.0 Schwäche: Padding

- ❑ M | MAC wird mit Padding auf ein Vielfaches der Blocklänge aufgefüllt (z.B. bei AES-128 max. 16 Byte); letztes Byte enthält die Länge
- ❑ Bei maximalem Padding Block steht hier 15
- ❑ MitM-Angreifer „sucht“ sich ein Byte aus, das er entschlüsseln möchte
- ❑ Byte wird ans Ende gestellt und ersetzt letztes Byte (Padding)
- ❑ Server entschlüsselt; Fallunterscheidung letztes Byte = 15
- ❑ Kommt in 1/256 Fällen vor
- ❑ Entschlüsselung letzter Block
- ❑ $M_n = D[C_n] \text{ XOR } [C_{n-1}]$ (wg. CBC)

■ Praktische Anwendung

- ❑ Angreifer erzeugt Script beim Client SSL-Sessions beim Server
- ❑ Versucht Session Cookie (Position ist bekannt) zu entschlüsseln
- ❑ Übernimmt dann SSL-Session

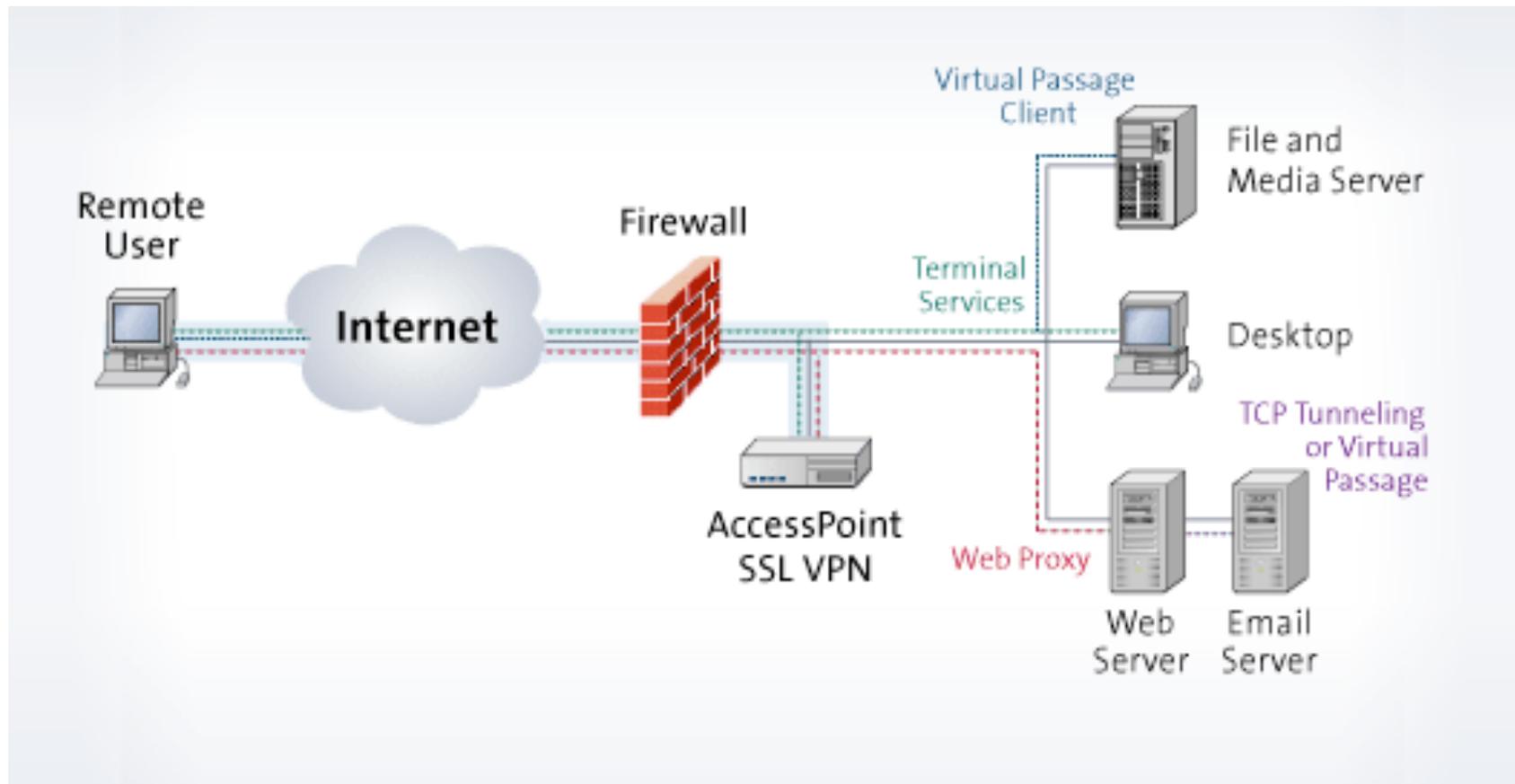
- Ziel wie z.B. bei PPTP und IPSec:
Transport geschützter Daten über öffentliche Netze

- Wiederum Site-to-Site- und End-to-End-Einsatz möglich

- Häufige Eigenschaften in der Praxis:
 - Nutzung von TCP-Port 443 (offene Firewall-Ports wg. HTTPS)
 - „Schlanker“ Client, z.T. ohne manuelle Software-Installation
 - Wichtig insb. für mobile Geräte (private Notebooks, Smartphones, ...)
 - Früher z.T. nur zur Nutzung von Webanwendungen geeignet (kein „richtiges“ VPN)
 - Start der VPN-Verbindung per Login auf einer Webseite
 - Zusätzliche Funktionalität per Browser-Plugin
 - z.B. Virens Scanner, Löschen des Cache bei Sitzungsende, ...

SSL-VPNs: Erste Generation

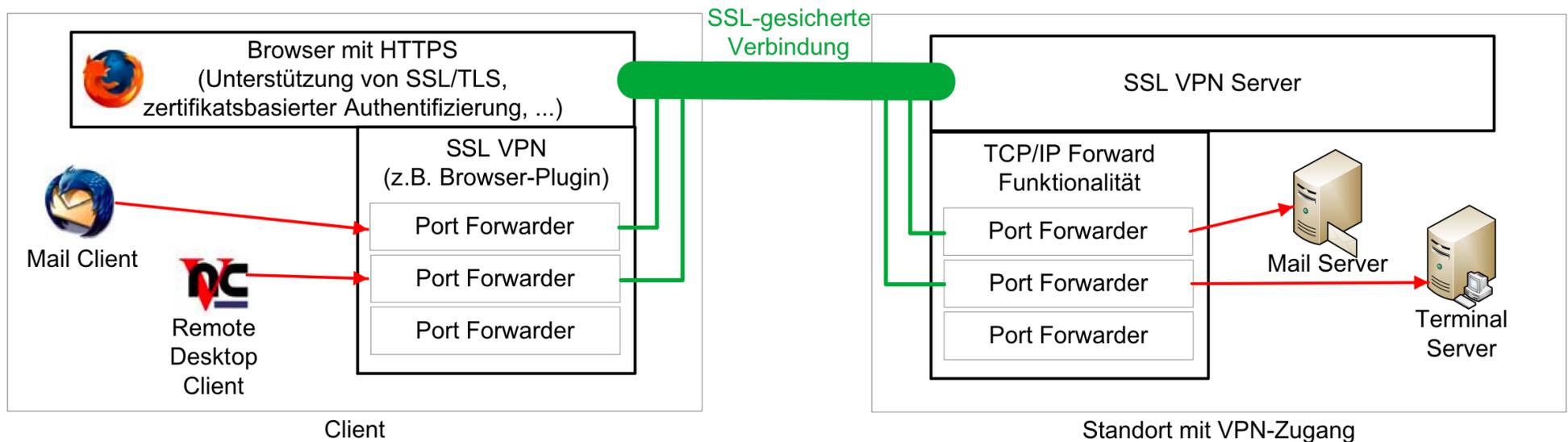
- Browserbasiert, einfache Inbetriebnahme
- Web Sessions für transparenten Client-IP-Adresswechsel (Roaming)



Bildquelle: www.menlogic.com

SSL-VPNs: Zweite Generation

- Web-basierte Installation eines „fat“ Clients
- Tunnel für TCP/IP-basierte Protokolle
- Z.T. Browser-Plugins für zusätzliche Funktionalität



- IPsec zur Standort-Vernetzung
 - ausgereifte Produkte (Appliances)
 - zertifizierte, standardbasierte Lösungen (Compliance)
- SSL-VPN für mobile Anwender
 - schlanker Client
 - unkomplizierte Installation durch Anwender
- Beispiel Cisco AnyConnect SSL VPN Client
(<https://asa-cluster.lrz.de>, nutzbar mit CampusLMU-Kennung)

 WebVPN Service

Login

Please enter your username and password.

GROUP:

USERNAME:

PASSWORD:

 Cisco AnyConnect VPN Client

<p>WebLaunch</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Platform Detection <ul style="list-style-type: none"> <input type="checkbox"/> - ActiveX <input type="checkbox"/> - Java Detection <input type="checkbox"/> - Sun Java <ul style="list-style-type: none"> <input type="checkbox"/> - Download <input type="checkbox"/> - Connected 	<p>Using Sun Java for Installation</p> <p>Launching Cisco AnyConnect VPN Client</p> <p>If the software does not start properly, Click here to end the session cleanly.</p>
---	---

VPN Client Downloader


 The installer is setting up the AnyConnect application. This could take a moment. Please wait...

 Cisco AnyConnect VPN Client

<p>WebLaunch</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Platform Detection <ul style="list-style-type: none"> <input type="checkbox"/> - ActiveX <input type="checkbox"/> - Java Detection <input checked="" type="checkbox"/> - Sun Java <ul style="list-style-type: none"> <input type="checkbox"/> - Download <input checked="" type="checkbox"/> - Connected 	<p>Connection Established</p> <p>The Cisco AnyConnect VPN Client has successfully connected.</p> <p>The connection can be controlled from the lock icon in your menu bar.</p> <div style="text-align: center; border: 1px solid #ccc; padding: 5px; width: fit-content; margin: 0 auto;">  </div> <p style="text-align: right;"><input type="button" value="Download"/></p>
---	---

Überprüfung von Web-Server auf SSL-Schwächen

■ <https://www.ssllabs.com/ssltest/>

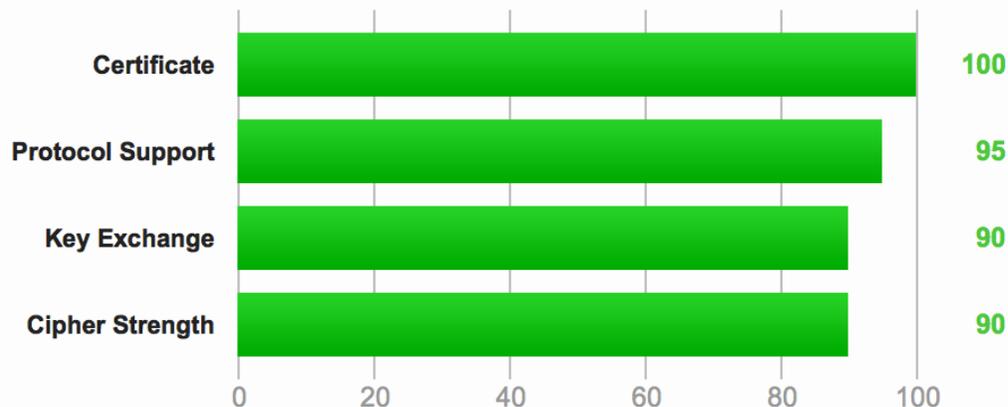
SSL Report: lrz.de (129.187.255.234)

Assessed on: Thu Jan 15 04:03:42 PST 2015 | [Clear cache](#)

[Scan Another »](#)

Summary

Overall Rating



Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

The server does not support Forward Secrecy with the reference browsers. Grade reduced to A-. [MORE INFO »](#)

This server supports TLS_FALLBACK_SCSV to prevent protocol downgrade attacks.



Zusatzfolien: Weitere SSL Vulnerabilities



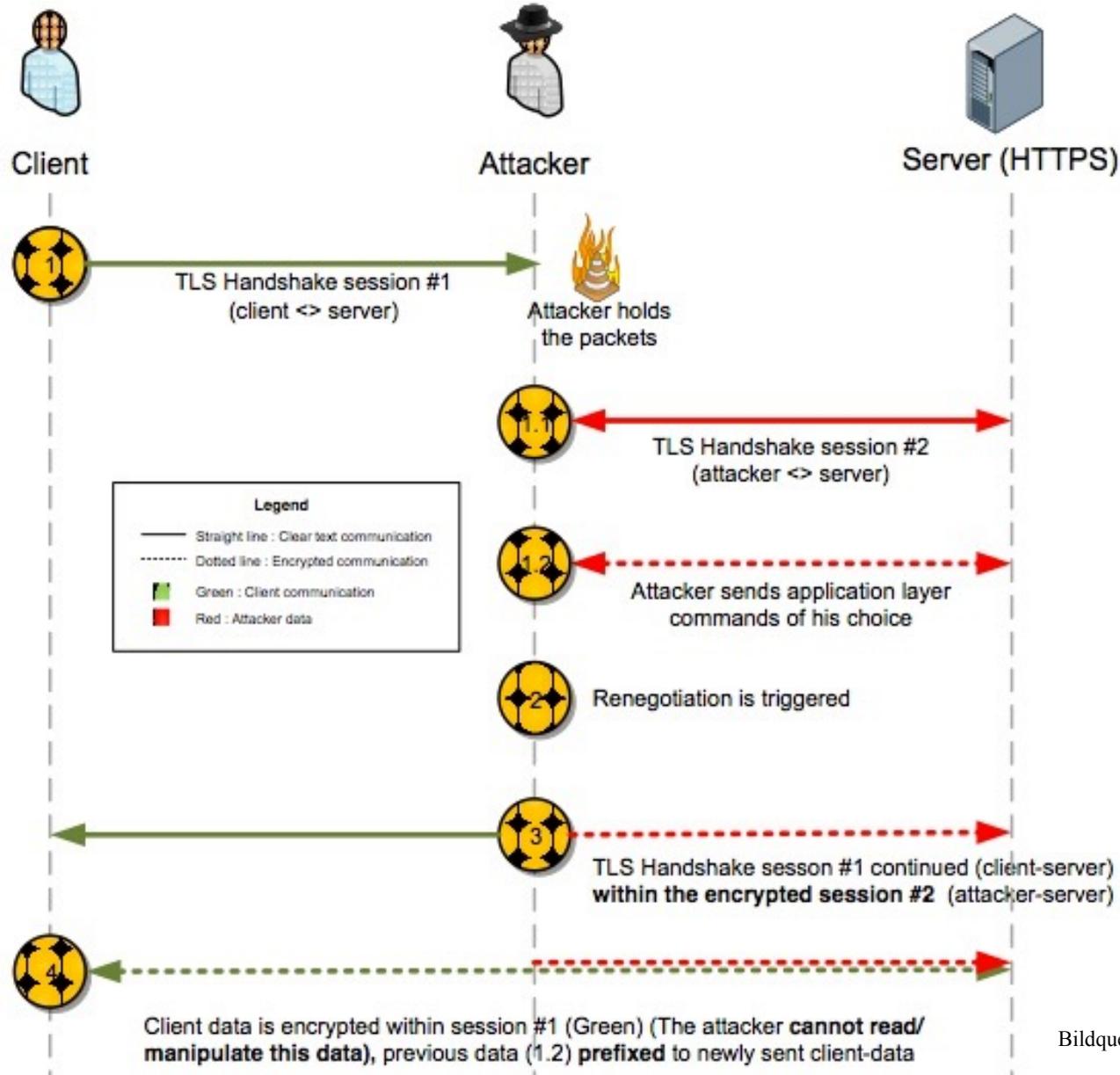
- Typische Ausgangssituation z.B. bei HTTPS:
 - Server kann erst nach Analyse der Client-Anfrage entscheiden, ob
 - Client-Zertifikat benötigt wird und/oder
 - stärkere kryptographische Verfahren eingesetzt werden sollen
 - Zu diesem Zeitpunkt besteht schon eine SSL-/TLS-gesicherte Verbindung ohne Client-Zertifikat bzw. mit zu schwachen Verfahren

- Server muss SSL-/TLS-Renegotiation anstoßen, um diesmal das Client-Zertifikat zu verlangen bzw. andere Algorithmen zu wählen.

- Nach erfolgreicher Renegotiation wird die ursprüngliche - nicht ausreichend abgesicherte - Client-Anfrage abgearbeitet.

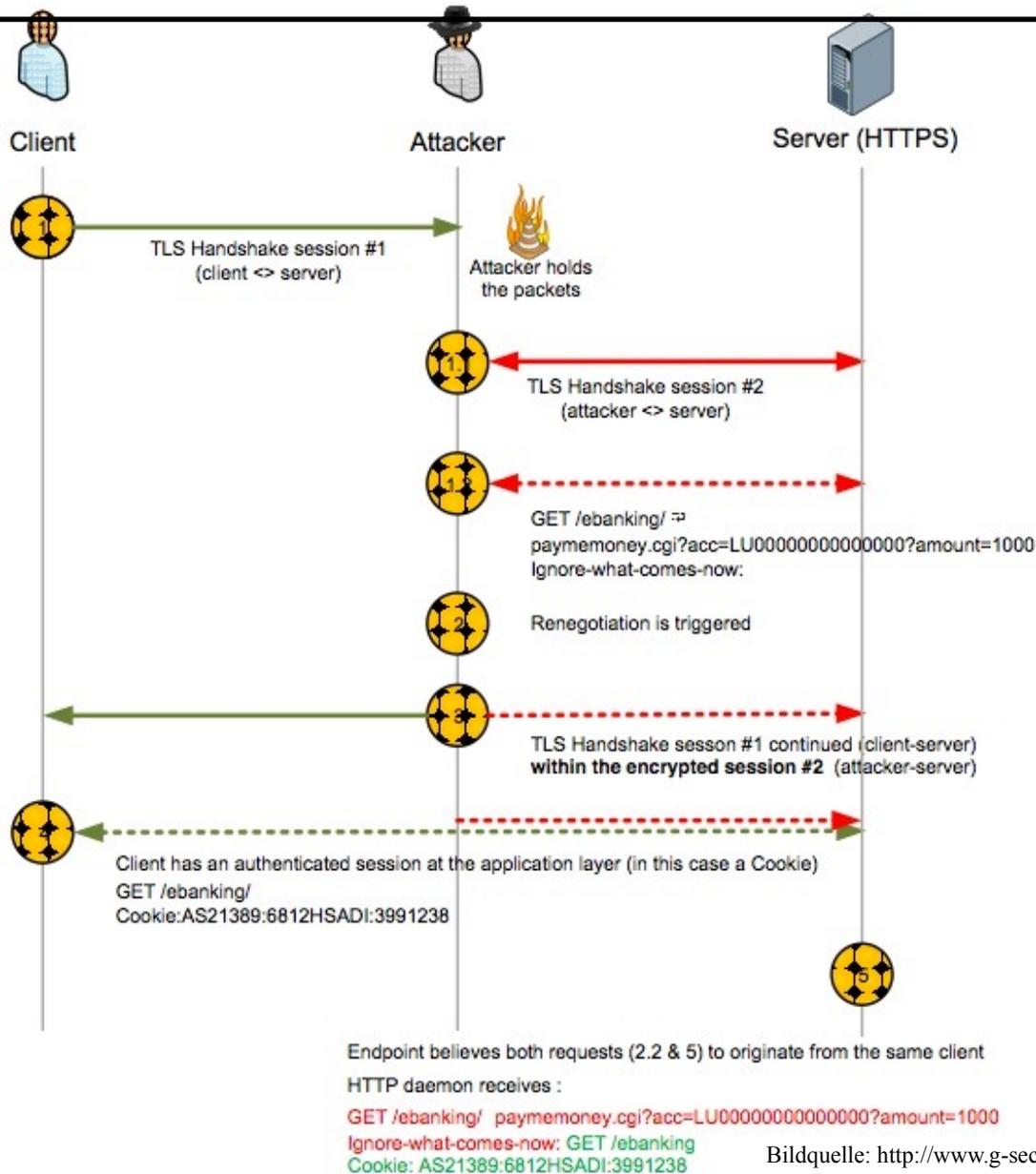
- „Man in the middle“ kann dies ausnutzen, um eigene Anfragen am Anfang der Verbindung einzuschleusen...

Einschub: Renegotiation attack - Allgemeiner Ablauf



Bildquelle: <http://www.g-sec.lu/practicaltls.pdf>

Einschub: Renegotiation attack - bei HTTPS



- Eigentliche Angreifer-Anfrage wird durch vorangestelltes „Ignore-what-comes-now:“ neutralisiert, aber später wirksam
- Dem Angreifer unbekanntes Autorisierungsdaten (z.B. Cookies) werden vom Client geliefert.
- Angreifer konkateniert Client und Angreifer-Nachrichten
- Am Ende glaubt der Server beide Nachrichten kommen vom selben Client (und belastet im Bsp. das Client-Konto)

Bildquelle: <http://www.g-sec.lu/practicaltls.pdf>

- **Betrifft nicht nur HTTPS**
 - Beispiel SMTP: Man-in-the-middle gibt Absender und Empfänger vor, erhält so z.B. die gesamte E-Mail samt Steuerkommandos vom Client

- **Workaround in der Phase zwischen Disclosure und Patch:**
 - Renegotiation per Konfiguration verbieten/deaktivieren
 - Problematisch, da inkompatibel zu den Anwendungen, die sie bislang nutzen

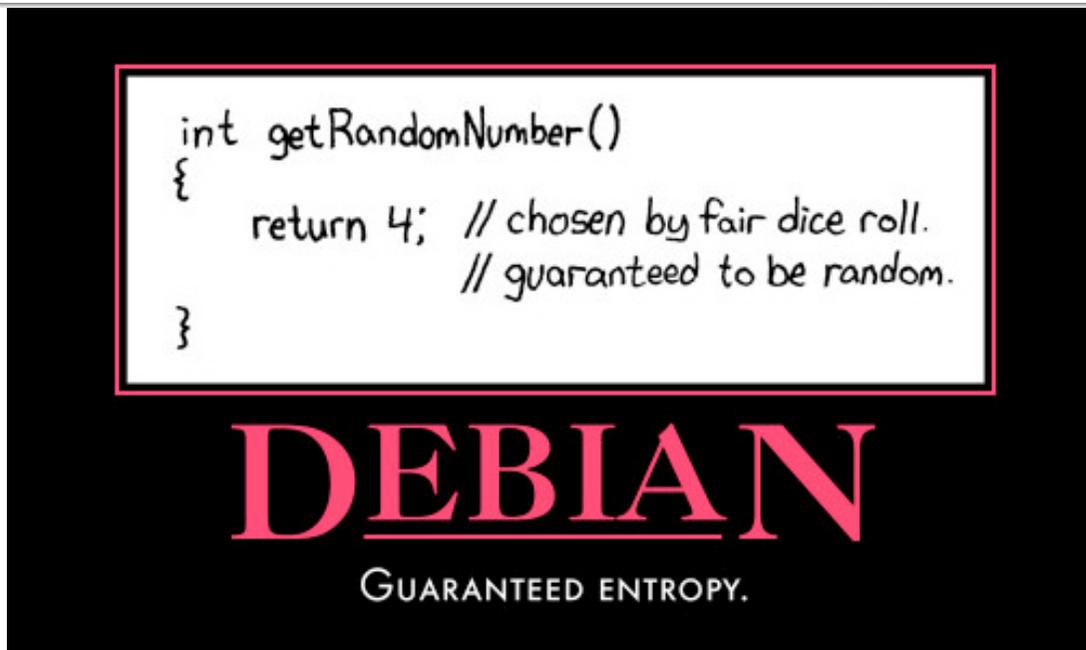
- **Lösungsansatz der IETF: TLS Extension**
 - TLS-Sessions werden kryptographisch an Clients gebunden
 - Clients werden über Renegotiation informiert
 - In der initialen Aushandlung kann die Möglichkeit zur Renegotiation deaktiviert werden
 - Wurde in vielen Implementierungen kurzfristig umgesetzt

- OpenSSL: Freie Bibliothek zur Implementierung von SSL / TLS
- Wird genutzt von vielen Linux Derivaten, OpenSSH, Apache (mod_ssl), Onion Router (TOR), OpenVPN, etc.
- 2006: Code Analyse-Tools Valgrind und Purify erkennen potentielle Schwachstellen im Quellcode
 - Uninitialisierter Speicher
 - wird gelesen bevor er geschrieben wird
- Maintainer berichtigen diesen „Bug“ mit einem Patch
 - zwei Zeilen werden gelöscht MD_Update(&m, buf, j)
- Code-Abschnitt wurde verwendet, um die Entropie des Zufallszahlengenerators (PRNG) zu verbessern
- Patch bewirkt das Entropie nur noch von Prozess-ID (PID) abhängt
- In vielen Linux-Systemen 15 Bit, d.h. Entropie von 2¹⁵

- Schlüsselraum von 32.767 Schlüsseln für jede Schlüssellänge und Schlüsseltyp
- 2008 von Luciano Belo (Argentinischer Forscher) entdeckt
- Erzeugung aller 1.024 und 2.048 Bit Schlüssel durch Moore:
 - Cluster mit 31 Xeon Cores
 - Zwei Stunden
 - Zusätzlich 6 Stunden für alle 4.096 Bit Schlüssel
- Betroffene Schlüssel:
 - SSH Host Keys
 - Benutzerschlüssel für Public Key Authentication in SSH
 - Sitzungsschlüssel
- SSH Perfect Forward Secrecy (PFS)
 - Debian Bug verhindert PFS
 - Selbst wenn Maschine sicheren Zufallszahlengenerator besitzt

- Schwacher Schlüssel kann einfach genutzt werden
 - `ssh -i weak-key4586 root@targetmachine`
 - Zertifikate sind einfach zu fälschen
- Damit Spoofing
 - SSL-gesicherte Web-Seiten
 - SSH
 - Bsp.: AKAMAI Server, verteilte
 - Elster (Software für Steuererklärungen)
 - Windows-Treiber von ATI für Grafikkarten
- Entschlüsselung des Verkehrs

- Alle Schlüsselpaare prüfen:
 - Skript (security.debian.org/project/extra/dowkd/dowkd.pl.gz)
- Firefox oder Internet Explorer Plugin zur Überprüfung
- Wiederruf von Zertifikaten mit schwachen Schlüsseln
 - Problem: CRLs oder OCSP wird oft nicht genutzt



Quelle: <http://trailofbits.files.wordpress.com/2008/07/hope-08-openssl.pdf>