Angriffe auf SSL 3.0 und TLS 1.0 bei Verwendung des CBC-Modes und Vorläuferversionen dieser Angriffe

Projektarbeit
Hochschule Karlsruhe - Technik und Wirtschaft

Tobias Wink

6. September 2019

Inhaltsverzeichnis

Akronyme

Bezeichnung	Beschreibung
BACPA	Blockwise-adaptive Chosen Plaintext Angriff
BCBA	Blockwise Chosen-Boundary Angriff
CBC	Cipher-Block-Chaining
CPA	Chosen Plaintext Angriff
IETF	Internet Engineering Task Force
IV	Initialisierungsvektor
MAC	Message Authentication Code
SSL	Secure Sockets Layer
TLS	Transport Layer Security

Symbolliste

Bezeichnung	Beschreibung
C_i	Bereits verschlüsselter Block
C_{i-1}	Verschlüsselter Vorgängerblock von C_i. Fungiert als Initialisierungsvektor bei der Verschlüsselung von C_i
C_j	Block, den der Angreifer als nächstes verschlüsseln möchte
C_{j-1}	Verschlüsselter Vorgängerblock von C_j. Fungiert als Initialisierungsvektor bei der Verschlüsselung von C_j
L	Die Blockgröße
e^{-1}	Die Entschlüsselungsoperation
e	Die Verschlüsselungsoperation
k	Der Schlüssel
P_i	Der Klartextblock von C_i
P_j	Der vom Angreifer gewählte Klartextblock, der zu C_j verschlüsselt wird
$\oplus$	Exklusiv-Oder (XOR)

Glossar

Blockchiffre

Blockchiffren bezeichnen Verschlüsselungsverfahren, bei denen ein Klartext in gleichgroße Blöcke aufgeteilt wird und diese Blöcke anschließend mit der selben Verschlüsselungsoperation mit dem identischen Schlüssel chiffriert werden [Arb15].

Message Authentication Code

Ein Message Authentication Code dient zur Sicherstellung der Integrität von Daten [Men96]. Er wird durch schlüsselabhängige kryptographische Hashfunktionen erzeugt.

Stromchiffre

Stromchiffren bezeichnen Verschlüsselungsverfahren, bei denen Klartextzeichen nacheinander mit einem in jedem Schritt variierenden Element eines Schlüsselstroms kombiniert werden [Arb15].

KAPITEL 1

Einleitung

Im Herbst 2014 haben die Google-Mitarbeiter Bodo Möller, Thai Duong und Krzysztof Kotowicz Details über POODLE veröffentlicht, einen Angriffsvektor, der das WWW erschüttern sollte. Obwohl zwei Kommunikationsteilnehmer das gleiche als sicher geltende Protokoll für die Transportverschlüsselung unterstützen, konnte es Angreifern gelingen, die eingesetzte Protokollversion der Parteien auf eine veraltete angreifbare Version herabzusetzen und dadurch unter anderem Login-Informationen auszulesen. Zu diesem Zeitpunkt waren etwa 61% der Cloud-Server weltweit davon bedroht [Sar14].

1.1 Ziel und Aufbau dieser Arbeit

Ziel dieser Arbeit ist die chronologische Aufarbeitung von Angriffen auf SSL 3.0 und TLS 1.0, die Schwächen des CBC-Modes ausnutzen, welche trotz vorhandener Nachfolgeversionen der Protokolle durch Abwärtskompatibilität in eingesetzter Software ein Sicherheitsrisiko darstellen.

Zu Beginn werden zunächst die für das Verständnis der Arbeit entscheidenden Grundlagen vermittelt.

Anschließend werden in chronologischer Reihenfolge die verschiedenen Angriffe erläutert. Dabei wird aufgezeigt, wie anfänglich noch unpraktikable und aufwändige Angriffsszenarien im Laufe der Zeit von verschiedenen Parteien weiter verfeinert wurden, bis sie in einem weitgreifenden realistischen Sicherheitsdebakel endeten.

Die jeweiligen Abschnitte beziehen sich dabei immer auf eine konkrete gefundene Schwäche:

Abschnitt 3.1: Wei Dai fand 2002 eine Lücke im SSH2-Protokoll, die den CBC-Mode von Blockchiffren betrifft.

Abschnitt 3.2: Gregory V. Bard wandte 2004 die Lücke von Wei Dai bei SSL 3.0 und TLS 1.0 an.

Abschnitt 3.3: Gregory V. Bard entwickelte 2006 sein Angriffsszenario weiter.

Abschnitt 3.4: Thai Duong und Juliano Rizzo verfeinerten 2011 die bisher bekannten Schwächen zu einer effizienten Attacke namens BEAST.

Abschnitt 3.5: Bodo Möller, Thai Duong und Krzysztof Kotowicz haben 2014 einen weiteren Angriffsvektor bei SSL 3.0 unter der Verwendung von Blockchiffren im CBC-Mode gefunden, welcher POODLE getauft wurde.

KAPITEL 2

Grundlagen

2.1 CBC-Mode

Bei der symmetrischen Verschlüsselung wird zwischen zwei Verfahren unterschieden: Stromchiffren und Blockchiffren. Beide Arten haben verfahrensbedingt unterschiedliche Stärken und Schwächen, die an dieser Stelle weder aufgezeigt noch diskutiert werden sollen, außer dass eine Blockchiffre per se immer nur einen einzigen Block verschlüsselt. Zum Zeitpunkt der hier gezeigten Angriffe lag die Blockgröße bei 64 Bit[1]. Bei der Anwendung von Blockchiffren auf Texte, deren Länge die jeweilige Blockgröße übersteigt, benötigt man daher eine Methode, um dieser Einschränkung zu begegnen. Solche Methoden nennt man Betriebsmodi und der CBC-Mode ist einer davon.

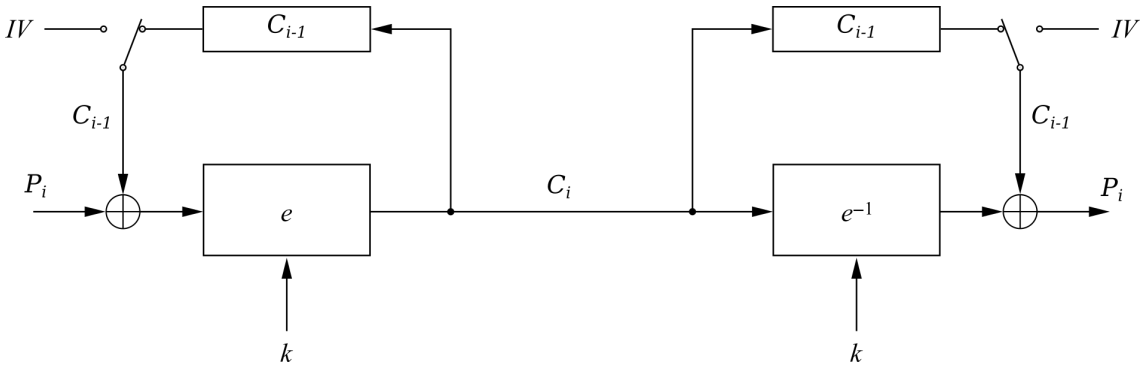

Abbildung 2.1: Ver- und Entschlüsselung im CBC-Mode, [Paa10]

CBC steht für Cipher-Block-Chaining, also für die Verkettung der verschlüsselten Blöcke, was bereits die Beschreibung des Verfahrens ist. Abbildung 2.1 zeigt eine schematische Darstellung des CBC-Modes, auf der linken Seite die Ver- und auf der rechten Seite die Entschlüsselung. Ein unverschlüsselter Block P_i wird mit dem verschlüsselten Vorgängerblock C_{i-1} XOR-verknüpft, anschließend mit Hilfe der Blockchiffre e verschlüsselt und danach neben der Übertragung wieder zurückgeführt, um mit dem nächsten unverschlüsselten Block XOR-verknüpft zu werden. Da es beim ersten Block noch keinen verschlüsselten Vorgängerblock gibt, wird dafür ein Initialisierungsvektor (IV) verwendet, welcher zuvor möglichst zufällig gewählt wurde. Bei der Entschlüsselung verläuft das Verfahren entsprechend umgekehrt. Zuerst wird entschlüsselt und anschließend wird durch die erneute

1 Die Blockgröße variiert heute üblicherweise zwischen 128 Bit und 256 Bit.

XOR-Verknüpfung mit dem verschlüsselten Vorgängerblock bzw. IV der unverschlüsselte Block wiederhergestellt.

Durch diese Verkettung soll unter anderem verhindert werden, dass zwei Blöcke, die denselben Klartext beinhalten, nach der Verschlüsselung weiterhin identisch sind.

Damit man den CBC-Mode benutzen kann, muss die Länge des zu verschlüsselnden Klartextes ein Vielfaches der Blockgröße sein. Ist dies nicht der Fall, so wird die Länge künstlich aufgefüllt. Dieser Vorgang nennt sich Padding und die verwendeten Zeichen heißen Paddingzeichen.

2.2 Chosen Plaintext Angriff (CPA)

Chosen Plaintext Angriffe beschreiben Angriffe, bei denen der Angreifer in der Lage ist, Klartexte frei zu wählen und diese verschlüsseln zu lassen [And08].

Um Angriffsszenarien verständlich zu beschreiben, werden oft Sicherheitsspiele eingesetzt. Dabei tritt üblicherweise ein Herausforderer gegen einen Angreifer an. Der Herausforderer, auch Orakel genannt, erstellt zu Beginn des Spiels eine verschlüsselte Nachricht und schickt sie dem Angreifer zu. Der Angreifer hat nun die Möglichkeit, dem Herausforderer Nachrichten zu schicken, die dieser dann verschlüsselt an den Angreifer zurückschickt. Der Angreifer darf sich so viele Nachrichten verschlüsseln lassen wie er möchte. Ziel des Angreifers ist es, nach möglichst wenigen Versuchen Aussagen über den Klartext der ersten Nachricht des Herausforderers zu treffen.

2.2.1 Blockwise-adaptive Chosen Plaintext Angriff (BACPA)

Die Definition von Chosen Plaintext Angriffen ist sehr allgemein gehalten. Und es wird oftmals davon ausgegangen, dass eine Nachricht, die verschlüsselt wird, atomar ist und nicht in kleinere Komponenten unterteilt werden kann. Dies ist in der Praxis, vor allem im Bereich der Netzwerkkommunikation, nur selten der Fall. Dort wird eine Nachricht in Pakete bzw. Blöcke unterteilt und sobald die Blöcke verschlüsselt wurden, werden sie versendet.

Um also ein Angriffsszenario zu beschreiben, bei dem ein Angreifer die Möglichkeit haben soll, auf das Ergebnis der Verschlüsselung eines Nachrichtenblocks zu reagieren anstatt auf die verschlüsselte ganze Nachricht, haben Antoine Joux et al. im Rahmen von [Jou02] den Blockwise-adaptive Chosen Plaintext Angriff eingeführt.

2.2.1.1 Blockwise Chosen-Boundary Angriff (BCBA)

Thai Duong und Juliano Rizzo haben im Rahmen von „Here come the ⊕ ninjas“ [Duo11b] den Blockwise Chosen-Boundary Angriff (BCBA) eingeführt, welcher eine weitere Verfeinerung des Blockwise-adaptive Chosen Plaintext Angriff (BACPA) ist. Eine Nachricht m besteht aus l Bytes $m[1], m[2], ..., m[l]$. Bei Verwendung einer Blockchiffre wird eine Nachricht m in s Blöcke der Größe b aufgeteilt $P_1, P_2, ..., P_s$. Die Trennung zwischen den Blöcken P_1 und P_2 findet nach $m[b]$ statt. Bei einem BCBA hat nun ein Angreifer die Möglichkeit, vor der Verschlüsselung die Blockgrenzen beliebig zu verschieben, beispielsweise durch das Voranstellen einer Zeichenkette vor m, bevor m verschlüsselt wird. Dies wird in Abbildung 2.2 anhand eines konkreten Beispiels illustriert.

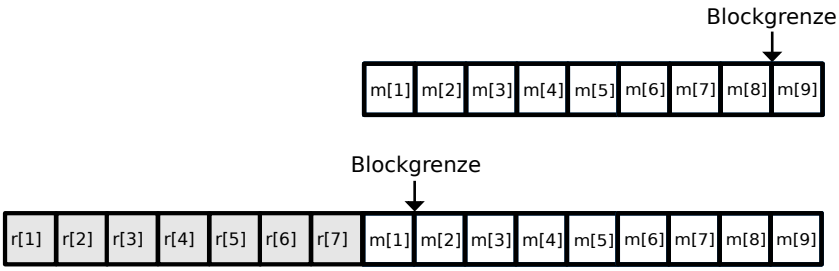

Abbildung 2.2: Verschieben der Blockgrenzen durch das Voranstellen einer Zeichenfolge, [Duo11b]

2.3 Ein kurzer Überblick über SSL und TLS

Das Secure Sockets Layer (SSL) Protokoll und das Transport Layer Security (TLS) Protokoll sind hybride Verschlüsselungsprotokolle zur sicheren Datenübertragung im Internet. SSL wurde ursprünglich von Netscape entwickelt und 1995 offiziell als SSL 2.0 das erste Mal veröffentlicht. Später wurde die Entwicklung von der Internet Engineering Task Force (IETF) übernommen und in diesem Zuge in TLS umbenannt. Eine Übersicht über die SSL-/TLS-Versionen zum Erstellzeitpunkt dieser Arbeit ist in Tabelle 2.1 zu sehen.

Tabelle 2.1: Übersicht über SSL-/TLS-Versionen

Version	Erscheinungsjahr	Verwendung nicht mehr empfohlen seit
SSL 2.0	1995	März 2011 [RFC6176]
SSL 3.0	1996 [RFC6101]	Juni 2015 [RFC7568]
TLS 1.0	1999 [RFC2246]	In Planung [Mor19]
TLS 1.1	2006 [RFC4346]	In Planung [Mor19]
TLS 1.2	2008 [RFC5246]	-
TLS 1.3	2018 [RFC8446]	-

Diese Arbeit beschreibt Angriffe auf SSL 3.0 und TLS 1.0, zwei Versionen die sich sehr ähnlich sind. Genauer heißt es im zugehörigen RFC [RFC2246] von TLS 1.0: „Die Unterschiede zwischen diesem Protokoll und SSL 3.0 sind nicht dramatisch, aber sie sind groß genug, dass TLS 1.0 und SSL 3.0 nicht zusammenarbeiten.“ Daher wird hier, solange keine spezielle Version gemeint ist, stellvertretend SSL verwendet, wenn SSL 3.0 und TLS 1.0 gemeint sind.

SSL kann mit verschiedenen Protokollen aus der Anwendungsebene (des OSI-Modells [ITU94]) kombiniert werden. SSL nimmt dabei Nachrichten entgegen, die übertragen werden sollen, teilt diese Daten in handhabbare Blöcke, komprimiert diese optional[1], erzeugt für jeden Block den zugehörigen Message Authentication Code (MAC), verschlüsselt nun den

1 Die in dieser Arbeit gezeigten Angriffe funktionieren nicht mit eingeschalteter Komprimierung. In der Praxis wird diese Verschlüsselung aber nur selten genutzt [Bar04].

Block und den MAC und sendet das Ergebnis an den Empfänger. Empfangene Daten werden entschlüsselt, verifiziert, optional entpackt, wieder zur gesamten Nachricht zusammengesetzt und anschließend an die höhere Ebene übergeben [RFC6101].

2.3.1 Initialisierungsvektoren bei SSL 3.0 und TLS 1.0

Die eigentliche Transportverschlüsselung beruht auf symmetrischer Verschlüsselung und findet innerhalb des Record Layers statt, wobei ein Record einer Nachricht zwischen Sender und Empfänger entspricht. Der Aufbau eines solchen Records sieht dabei etwa so aus [Bar04]:

- Unverschlüsselt:
  - Nachrichtentyp (1 Byte)
  - Versionsnummer (2 Bytes)
  - Nachrichtenlänge (2 Bytes)
- Verschlüsselt:
  - Nutzdaten (Beliebige Länge $\leq 2^{14}$ Bytes)
  - Message Authentication Code (typischerweise 20 Bytes)
  - Paddingzeichen (0 - Blockgröße − 1 Bytes; Es muss sichergestellt werden, dass die Länge des Klartextes ein Vielfaches der Blockgröße ist)
  - Paddinglänge (Anzahl der Paddingzeichen) (1 Byte)

Ein, wie man später erfahren wird, entscheidender Unterschied zwischen SSL 3.0 und TLS 1.0 ist, dass bei TLS 1.0 die Paddingzeichen dem Wert der Paddinglänge entsprechen müssen.

Zu den unterstützten Verschlüsselungsalgorithmen gehören auch Blockchiffren im Cipher-Block-Chaining (CBC)-Mode. Dabei wird für den Einsatz von Blockchiffren im CBC-Mode von beiden Protokollen definiert, dass der initiale IV im Rahmen des Handshakes festgelegt wird. Für jeden weiteren IV wird der letzte Block der vorherigen Nachricht verwendet [RFC2246; RFC6101]. Das bedeutet, dass alle Nachrichten einem einzelnen Strom von verschlüsselten Blöcken entsprechen.

2.4 Ein kurzer Überblick über HTTP(S)

Das Hypertext Transfer Protocol (HTTP) ist ein Protokoll der Anwendungsebene (des OSI-Modells [ITU94]) für verteilte kollaborative Hypermedien-Informationssysteme [RFC2616]. Es bildet die Grundlage des WWWs.

HTTP ist bis einschließlich Version 1.1 textbasiert, als Zeilenumbruch ist `<CR><LF>` (0x0C,0x0A) vorgeschrieben.

Eine HTTP-Anfrage ist allgemein wie folgt aufgebaut:

- Anfragenzeile bestehend aus der Zugriffsmethode, der URI der gewünschten Ressource und der eingesetzten HTTP-Version, z.B. `GET /index.html HTTP/1.1`
- Kopfzeilen, z.B. ein Cookie-Eintrag: `sessionid=da82c63dfb9e563904c0c08527c61fc0;.`

- Eine Leerzeile
- Der optionale Nachrichtentext

Ein Beispiel einer solchen Anfrage ist in Listing 2.1 dargestellt.

```
1 GET / HTTP/1.1
2 Host: www.google.de
3 User-Agent: curl/7.65.1
4 Accept: */*
5
```

Listing 2.1: Eine GET-Anfrage bei www.google.de via cURL

Das HTTPS-Protokoll ist streng genommen kein eigenständiges Protokoll. Es ist die Kombination aus HTTP und SSL. Die Nachrichten zwischen Server und Client sind normale HTTP-Nachrichten, die für den Transport mit Hilfe von SSL verschlüsselt werden. Dadurch ist einem Beobachter zwar bekannt, welche Parteien miteinander kommunizieren, aber nicht was, da der Inhalt, einschließlich Kopfzeilen und Ressource, durch die Transportverschlüsselung geschützt ist. Daher wird es vor allem in sicherheitskritischen Situationen eingesetzt wie z.B. Onlinebanking.

2.5 Ein vereinfachter Vergleich zwischen SSL, SSH und IPsec

SSL, SSH und IPsec sind alles Protokolle, die einen sicheren Kanal zwischen zwei Gesprächspartnern herstellen wollen. Alle Protokolle authentifizieren (via MACs) und verschlüsseln die Nachrichten symmetrisch. Die Reihenfolge von Authentifizierung und Verschlüsselung variiert aber zwischen den Protokollen (Abbildung 2.3).

SSL: $a = Auth(x)$, $C = Enc(x,a)$, $transmit\ C$
IPsec: $C = Enc(x)$, $a = Auth(C)$, $transmit\ (C,a)$
SSH: $C = Enc(x)$, $a = Auth(x)$, $transmit\ (C,a)$

Abbildung 2.3: Methoden für Authentifizierung und Verschlüsselung von SSL, SSH und IPsec [Kra01]

Da alle drei Protokolle auch Blockchiffren im CBC-Mode zur symmetrischen Verschlüsselung unterstützen, können gefundene Lücken, den CBC-Mode betreffend, unter Umständen von einem der Protokolle auch auf die anderen Protokolle übertragen werden.

KAPITEL 3

Chronologische Aufarbeitung der Angriffe

3.1 2002 - Eine Schwäche im SSH2-Protokoll

2002 wurde auf der Mailingliste der SSH-Arbeitsgruppe der IETF von Wei Dai ein Angriff auf das SSH2-Protokoll veröffentlicht [Dai02]. Er zitiert dabei aus Phil Rogaways Analyse des IPsec-Protokolls [Rog95], dass der CBC-Mode anfällig für Chosen Plaintext Angriffe (CPAs) ist, wenn dem Angreifer der IV bekannt ist oder er von ihm vorhergesagt werden kann, bevor dieser den Klartext wählt.

Wei Dai erweitert diese Aussage dahingehend, dass ein CPA auf den CBC-Mode möglich ist, wenn der letzte verschlüsselte Block bekannt ist.

Sollte es einem Angreifer gelingen, die Kommunikation abzuhören und sollte er weiterhin in der Lage sein, einen CPA durchzuführen, kann er überprüfen, ob ein bereits verschlüsselter Block einen bestimmten Wert als Klartext enthält.

3.1.1 Der Angriff

Wenn überprüft werden soll, ob der verschlüsselte Block C_i den Klartext P_i enthält, kann ein Angreifer seinen Klartextblock P_j so geschickt wählen, dass er den Vorteil vom CBC-Mode, dass für identische Klartexte unterschiedliche Ciphertexte generiert werden, wieder aufheben kann.

Dafür wird P_j=$P_i \oplus C_{i-1} \oplus C_{j-1}$ gewählt und zu C_j verschlüsselt. C_{i-1} ist der Vorgängerblock von C_i, der als IV von C_i verwendet wurde und C_{j-1} ist der Vorgängerblock von C_j, der als IV bei der Verschlüsselung von P_j verwendet wird.

Sollte P_i wirklich dem Chiffretext von C_i entsprechen, so gilt:

$$C_i = e(P_i \oplus C_{i-1})$$

$$\begin{aligned} C_j &= e(P_j \oplus C_{j-1}) \\ &= e(P_i \oplus C_{i-1} \oplus C_{j-1} \oplus C_{j-1}) \\ &= e(P_i \oplus C_{i-1}) \\ &= C_i \end{aligned}$$

Somit ist ein Angreifer durch das Wissen, dass $IV_{C_i} = C_{i-1}$ und $IV_{C_j} = C_{j-1}$ gilt, in der Lage, bei der Wahl von P_j die Werte der IVs herauszurechnen, um dadurch Werte für P_i zu raten, vorausgesetzt er kennt C_{i-1}, C_i, C_{j-1} und C_j.

Die Schwierigkeit für einen Angreifer, diese Schwäche in der Praxis auszunutzen, liegt in den folgenden Herausforderungen:

1. Die Möglichkeit, einen CPA durchführen zu können.
2. Das Wissen über Nachrichteninhalte, die womöglich kommuniziert wurden.

Im Verlauf der Konversation des Threads wird von Wei Dai ausgesagt, dass diese Schwäche bereits für IPsec bekannt war und er nicht ausschließen möchte, dass es ebenfalls für SSL gilt. Bodo Möller bestätigt darauf, dass diese Schwäche ebenfalls für TLS 1.0 gelten muss, da, wie in Unterabschnitt 2.3.1 erläutert, bei SSL der letzte Block der vorherigen Nachricht als impliziter IV verwendet wird.

3.2 2004 - Ein sehr unpraktikabler Chosen Plaintext Angriff gegen das SSL-Protokoll

2004 hat Gregory V. Bard mit „The Vulnerability of SSL to Chosen Plaintext Attack“ [Bar04] gezeigt, dass sich die gefundene Schwachstelle bei SSH2 wirklich auf SSL anwenden lässt und Bodo Möller mit seiner Aussage somit recht hatte [Dai02]. Der Angriff basiert auf der gleichen Erkenntnis wie der von Wei Dai, dass das Wissen über IVs für CPAs genutzt werden kann.

Der Mehrwert dieser Arbeit liegt nicht nur im Transfer des Angriffs von SSH2 auf SSL, sondern ebenfalls in den Erkenntnissen über die praktische Umsetzbarkeit. Es wird ein sehr unpraktikables, aber durchaus mögliches, Angriffsszenario beschrieben, welches Lösungen für die in Abschnitt 3.1.1 genannten Herausforderungen liefert, die nachfolgend genauer betrachtet werden:

1. Der CPA wird über ein vom Angreifer erstelltes Browserplugin durchgeführt.
2. Die für den Angreifer interessanten Nachrichteninhalte sind PINs und kurze Passwörter, sodass die Entropie relativ gering ist.

3.2.1 Das Browserplugin

Damit ein Angreifer mit einem präparierten Browserplugin einen solchen Angriff ausführen kann, muss das Opfer zuerst das Plugin installiert haben. Ein möglicher Infektionsweg, der von Bard beschrieben wird [Bar04], ist es, eine präparierte Webseite zu erstellen, welche ein Objekt mit einem vom Browser nicht unterstützten Datentyp anzeigen möchte. Dadurch wird das Opfer aufgefordert, ein Plugin zu installieren, welches den Datentyp unterstützt. Die Information, dass das schadhafte Plugin diesen Datentyp anzeigen kann, lässt sich auf der Webseite hinterlegen.

Bereits 2004 hatten Browserplugins im Allgemeinen keinen Zugriff auf Formulardaten. Abbildung 3.1 veranschaulicht die verschiedenen Kommunikationswege im Browser. Die Formulardaten wandern vom Betriebssystem über die Pfade $E \rightarrow C \rightarrow D$ zum „Session Layer“, wo sie verschlüsselt werden, um anschließend mit Hilfe des „Transport Layers“ durch das Internet zum Server übertragen zu werden, während einem Browserplugin nur die Kommunikationswege A und B zur Verfügung stehen. Weiterhin kann man erkennen, dass ein Browserplugin über A die Möglichkeit hat, eigene Nachrichten zu verschlüsseln sowie zu verschicken und somit generell die Möglichkeit zum BACPA besitzt.

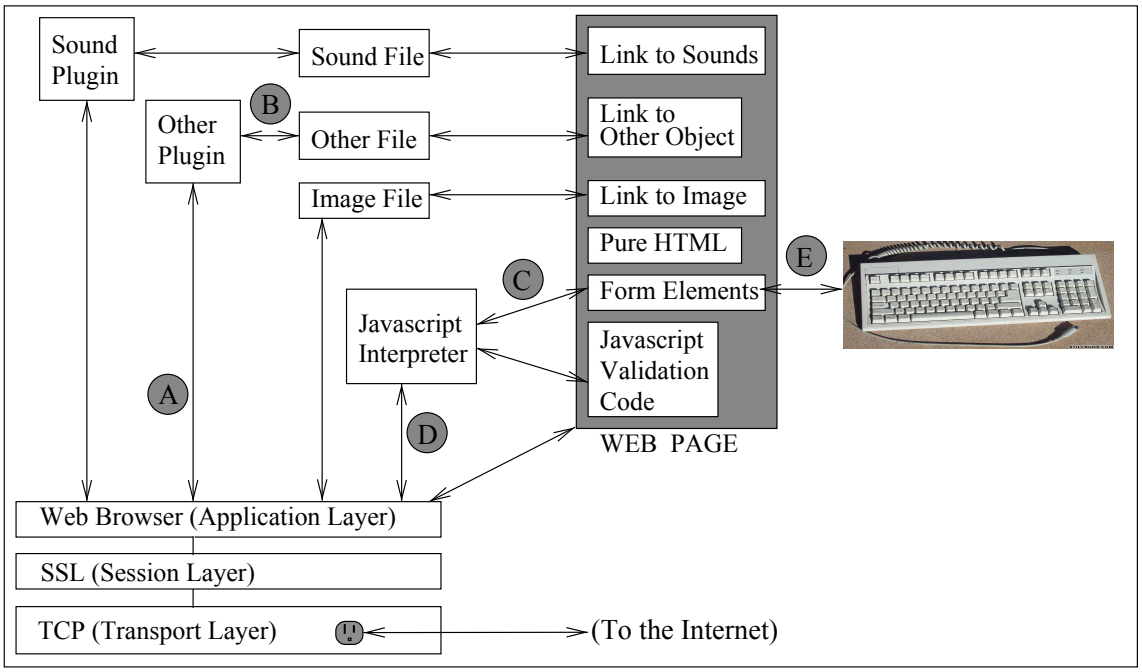

Abbildung 3.1: Datenfluss im Browser eines Formulars mit aktivem SSL, [Bar04]

Um das gezeigte Angriffsszenario umsetzen zu können, benötigt das Browserplugin aber ebenfalls Zugriff auf die bereits verschlüsselten Nachrichtenblöcke. Dass das Abfangen der Nachrichtenblöcke möglich ist, sieht Bard darin begründet, dass SSL eingesetzt wird und daher der Transportweg als unsicher angesehen wird. Für den Rückkanal werden ebenfalls verschiedene Ideen vorgeschlagen:

1. Das Browserplugin hat direkten Zugriff auf ein Werkzeug (bspw. *tcpdump*), um den Nachrichtenverkehr selbst mitzuschneiden.
2. Der Angreifer befindet sich im selben Netzwerk und schafft es den verschlüsselten Nachrichtenverkehr des Opfers mitzuschneiden, um dem schadhaften Plugin die notwendigen Informationen zukommen zu lassen, z.B. indem der Angreifer die mitgeschnittenen Daten per Webserver dem Plugin zum Abruf zur Verfügung stellt.

Es lässt sich erahnen, dass sich dieses Angriffsszenario in der Praxis nur mit hohem Aufwand realisieren lässt, da das Opfer nicht nur zur Installation des vorbereiteten Plugins gebracht werden muss, sondern weiterhin dem Browserplugin die richtigen Nachrichtenblöcke zur Verfügung stehen müssen, um einen Angriff durchführen zu können.

3.2.2 Das Wissen über Nachrichteninhalte

Um den Angriff erfolgreich umsetzen zu können, muss ein Angreifer in der Lage sein, den Klartext P_i des zu erratenen Nachrichtenblocks genau zu reproduzieren. Das bedeutet, er muss einerseits wissen, welchen Block er angreifen möchte und andererseits will er den Inhalt des Blocks mit möglichst wenigen Versuchen erraten.

Generell muss ein Angreifer also wissen bzw. vorhersagen können, welche Inhalte in welcher Reihenfolge vom Opfer an den Server verschickt werden. Da der Angriff mit Hilfe eines Browserplugins umgesetzt werden soll, wird versucht, Eingabedaten des Opfers auf einer Webseite zu erraten. Unter der Annahme, dass die Webseite, von welcher der Angreifer die Eingabedaten erraten möchte, generell für ihn erreichbar ist, kann er den Quelltext dieser Webseite studieren und dadurch bereits Teile der Kommunikation vorhersagen.

Wenn es sich zum Beispiel bei den Eingabedaten um eine PIN-Eingabe in ein HTML-Formular handelt, welches via `POST` an den Server geschickt werden soll, dann könnte der HTML-Code aussehen wie in Listing 3.1 gezeigt.

```
<form action="/login" method="post">
  <input type="hidden" name="name" value="inputForm">
  PIN: <input type="password" name="pinValue" minlength="4" maxlength="4"><br>
  <input type="submit" value="Login">
</form>
```

Listing 3.1: HTML-Code für ein einfaches Loginformular

Wenn dieses Formular mit der PIN *7633* abgeschickt wird, entsteht ein `POST`-Request vom Browser an den Server, welcher unverschlüsselt in Listing 3.2 zu sehen ist.

```
POST /login HTTP/1.1
Host: foo.example
Content-Type: application/x-www-form-urlencoded
Content-Length: 28

name=inputForm&pinValue=7633
```

Listing 3.2: Unverschlüsselter POST-Request

Wie zu erkennen ist, kann ein Angreifer einen Großteil dieser Informationen vorhersehen. Der einzige variable Wert ist die PIN *7633*. In diesem Beispiel handelt es sich immer um eine 4-stellige PIN, sodass der Wert für „Content-Length“ konstant ist, wobei der genaue Wert dafür für einen Angreifer irrelevant ist, da ihm nur bekannt sein muss, wie viele Stellen die Zahl hat, um die Blöcke richtig berechnen zu können.

Die Klartextblöcke einer Nachricht waren typischerweise 2^{14} Bytes lang und da ein einzelnes Zeichen normalerweise 1 Byte benötigte, bestand ein Block vor der Verschlüsselung aus maximal 14 Zeichen. Teilt man den Inhalt des POST-Requests aus Listing 3.2 in solche Blöcke, entstehen Klartextblöcke wie:

P_{i-1}: name=inputForm

P_i: &pinValue=7633

Man kann erkennen, dass der Angreifer von Block P_i 10 der 14 Werte kennt und nur die 4-stellige PIN raten muss. Da PINs numerisch sind und der Wertebereich dadurch auf 0-9 beschränkt ist, wurde die Anzahl an Möglichkeiten auf maximal $10^4 = 10000$ Möglichkeiten reduziert, wobei durchschnittlich 5000 Versuche für das Erraten ausreichen. Wenn der

gesamte Inhalt des Blocks unbekannt gewesen wäre, gäbe es $8 * 2^{14} = 2^{17} = 131072$ Möglichkeiten.

3.2.3 Mögliche Lösungen

Es werden zwei Möglichkeiten aufgezeigt, wie diese Angriffe in einer zukünftigen Protokollversion verhindert werden können:

1. Die Verwendung von expliziten (pseudo-)zufälligen IVs für jede Nachricht, die verschickt wird.
2. Die Verwendung eines anderen Betriebsmodus.

3.3 2006 - Ein unpraktikabler Blockwise-adaptive Chosen Plaintext Angriff gegen das SSL-Protokoll

2006 hat Gregory V. Bard mit „A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL“ [Bar06] eine weitere Ausarbeitung zu diesem Thema veröffentlicht. An seiner Grundidee, Informationen mit geringer Entropie zu erraten, hat sich nichts geändert, dafür ist nun der CPA mit BACPA genauer spezifiziert (siehe Abschnitt 2.2.1).

Das gezeigte Angriffszenario basiert nun auch nicht mehr auf dem in Abschnitt 3.2.1 erläuterten Browserplugin, sondern auf einem Java-Applet.

Java-Applets sind Java-Programme, die in einer Sandbox-Umgebung innerhalb der Java Virtual Machine des Browsers ausgeführt werden. Durch diese geschützte Umgebung ist es für ein Applet nur dann möglich, Tastatureingaben mitzulesen, wenn es gerade vom Benutzer fokussiert ist. Ein Java-Applet besitzt aber das Recht, eine TCP-Verbindung zum Server aufzubauen, von dem es geladen wurde. Normalerweise sollte es dabei keine Missbrauchsmöglichkeit geben, da diese Verbindung losgelöst von anderen Verbindungen des Benutzers sein sollte, zum Beispiel von seiner Online-Banking Webseite in einem anderen Browsertab. Wenn der Benutzer nun aber einen SSL-Tunnel in Form eines HTTP-Proxies oder VPNs verwendet, gibt es für jedwede HTTP-Kommunikation nur eine Verbindung, die zum Tunnel. Damit gelten nun wieder vergleichbare Voraussetzungen wie im Angriff via Browserplugin und es besteht die Möglichkeit, Nachrichtenblöcke mit Hilfe des Java-Applets zu erraten.

Es besteht aber ebenso die Notwendigkeit eines Rückkanals, damit das Java-Applet die richtigen Nachrichtenblöcke verschicken kann und des Wissens über die Nachrichteninhalte (siehe Abschnitt 3.2.2).

Die Erstellung eines solchen Java-Applets sollte nicht aufwändiger sein als die Erstellung eines schadhaften Browserplugins. Weiterhin sollte es leichter sein, ein Opfer auf eine Webseite mit dem präparierten Applet zu locken, z.B. mit Hilfe einer Phishing E-Mail, als dass man es dazu bewegt, ein präpariertes Browserplugin zu installieren. Für den Rückkanal und das Wissen über die Nachrichteninhalte gelten ebenfalls dieselben Herausforderungen. Aber das hier erläuterte Angriffsszenario benötigt, im Gegensatz zum Angriff mit Hilfe des Browserplugins, ein Opfer, welches einen aktiven SSL-Tunnel für HTTP-Verbindungen hat.

3.3.1 Mögliche Lösungen

Zum Zeitpunkt der Veröffentlichung der Ausarbeitung von Bard 2006 befand sich TLS 1.1 noch in der Entwurfsphase und die Erkenntnisse über die Angreifbarkeit von Blockchiffren

im CBC-Mode durch das Vorhersagen der verwendeten IVs wurden berücksichtigt. Das bedeutet, dass TLS 1.1 nicht für solche Angriffe anfällig ist. Um diese Immunität zu erreichen, wurde das Protokoll so angepasst, dass nun explizite IVs verwendet werden. Diese expliziten IVs sind (Pseudo-)Zufallszahlen, welche immer als erster Block übertragen werden. Dadurch gibt es nun immer einen verschlüsselten Block mehr, als es Klartextblöcke gibt und einem Angreifer wurde die Möglichkeit genommen, den verwendeten IV vorherzusagen und somit wird das Angriffsszenario unmöglich.

Eine anderes Verfahren, welches alternativ von TLS 1.1 unterstützt wird, nennt sich „Single Block Nonces“. Dabei wird als Klartext des ersten Blocks einer Nachricht, welcher als IVs verwendet wird, immer eine Zeichenfolge genommen, die nur aus Nullen besteht. Das bedeutet, dass der IV, den der Angreifer für einen erfolgreichen Angriff vorhersehen muss, $IV = C_{i-1} = e(C_{i-2} \oplus 0000...0) = e(C_{i-2})$ lautet. Unter der Annahme, dass das Verschlüsselungsverfahren e sicher ist und der Angreifer den Schlüssel nicht kennt, ist die Wahrscheinlichkeit, dass der Angreifer den richtigen IV rät, genauso hoch wie die eines erfolgreichen Brute-Force Angriffs.

Eine Variation dieser Idee wird von OpenSSL ab Version 0.9.6d (09. März 2002) unterstützt. Dabei wird als erster Block einer Nachricht ein leerer Klartextblock erzeugt, der Inhalt besteht also ausschließlich aus dem MAC und den Padding-Informationen. Dieser „leere“ Block wird dann entsprechend verschlüsselt und fungiert als IV für den Rest der Nachricht. Diese Variante ist mit SSL kompatibel, wenn die Gegenseite mit dem leeren Block umgehen kann. Die Macher von OpenSSL gehen davon aus, dass durch dieses Verfahren der Angriffsvektor beseitigt wurde[1].

Es hat sich herausgestellt, dass es weitverbreitete Software gab, in erster Linie den Microsoft Internet Explorer, die mit dieser Änderung inkompatibel war [Möl04], sodass mit der Nachfolgeversion von OpenSSL (0.9.6e) eine Option hinzugefügt wurde, diese Gegenmaßnahme zu deaktivieren. Üblicherweise wurde diese Option dann auch verwendet, um in den Anwendungen die Gegenmaßnahme zu deaktivieren, um so eine höhere Kompatibilität zu erzielen.

3.4 2011 - Ein praktikabler Blockwise Chosen-Boundary Angriff gegen das SSL-Protokoll

2011 haben Thai Duong und Juliano Rizzo mit „Here come the $\oplus$ ninjas“ [Duo11b] den Angriff gegen SSL bei Verwendung von Blockchiffren im CBC-Mode auf ein neues Niveau gehoben. Dieser Angriff ist auch unter dem Akronym BEAST (Browser Exploit Against SSL/TLS) bekannt geworden.

Der Angriff baut auf den Vorarbeiten von Dai (Abschnitt 3.1) und Bard (Abschnitte 3.2 und 3.3) auf und setzt dabei auf einen BCBA (Abschnitt 2.2.1.1), welcher im Rahmen der Ausarbeitung ebenfalls definiert wurde. Abbildung 3.2 zeigt ein Sicherheitsspiel. Wie andere Spiele dieser Art besteht es aus einem Herausforderer und einem Angreifer und das Ziel ist, dass der Angreifer es schafft, die Nachricht m zu entschlüsseln. Der Herausforderer ist ein CBC-Verschlüsselungsorakel, welches während des Startens einen zufälligen Schlüssel

1 Der Hashwert eines leeren Blocks ist ein konstanter Wert und sollte somit ebenfalls von einem Angreifer „vorhergesagt“ werden können. Aus diesem Grund dürfte dieser Mechanismus nicht den gewünschten Mehrwert liefern, aber leider habe ich keine Erklärungen zu diesem Thema gefunden.

$k \in K$ und eine zufällige Nachricht $m \in M$ erzeugt. Der Angreifer seinerseits kann von einem Block zum nächsten innerhalb einer Nachricht adaptiv reagieren und ihm ist erlaubt, Zeichenketten beliebiger Länge m voranzustellen, bevor m verschlüsselt wird. Um dem Angreifer diese Freiheit zu gewähren, besteht dieses Sicherheitsspiel aus zwei Phasen:

Phase 1 (Chosen-Boundary): Der Angreifer generiert eine zufällige Zeichenkette r und sendet r an den Herausforderer. Der Herausforderer wählt einen zufälligen IV, stellt r m voran, verschlüsselt das Ergebnis dieser Konkatenation inklusive des Paddings im CBC-Mode mit Hilfe des Schlüssels k und des gewählten IVs, in der Abbildung mit C_0^* bezeichnet. Anschließend wird das verschlüsselte Ergebnis (C_0^*,C_1^*,C_1^*,...,C_s^*) zum Angreifer gesendet. Durch die Wahl von r kann die Blockgrenze beliebig verschoben werden.

Phase 2 (Blockwise): Der Angreifer sucht sich einen Block P_i aus und sendet diesen an das Orakel. Das Orakel verschlüsselt P_i mit Hilfe des Schlüssels k und dem letzten verschlüsselten Block als IV. Anschließend wird das verschlüsselte Ergebnis an den Angreifer gesendet. Der Angreifer wiederholt diese Phase so oft er möchte.

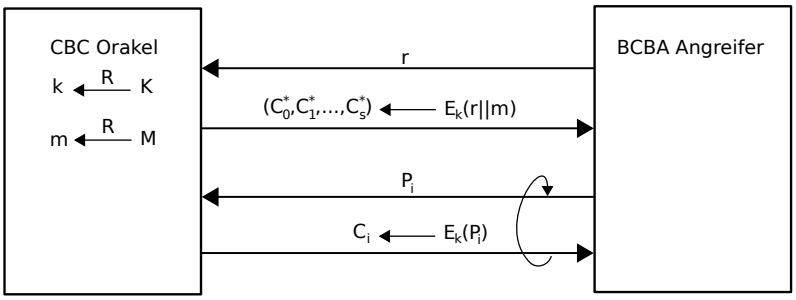

Abbildung 3.2: Ein Sicherheitsspiel zwischen BCBA und CBC-Mode, [Duo11b]

Es stellt sich heraus, dass der Angreifer das erste Bit $m[1]$ (Abbildung 2.2) spätestens nach 256 Wiederholungen von Phase 2 erraten hat. Mit viel Glück (Chance 1:256) schafft er es nach dem ersten Versuch, sodass er im Mittel etwa 128 Versuche benötigt. Um das zu erreichen, geht der Angreifer folgendermaßen vor:

1. Generiere eine zufällige Zeichenkette r mit $L-1$ Bytes, wobei L der Blockgröße eines Nachrichtenblocks entspricht und übertrage r an den Herausforderer (Phase 1).
2. Der Herausforderer stellt r m voran, füllt das Ergebnis mit entsprechend vielen Paddingzeichen auf, teilt die gesamte Nachricht in s-viele Blöcke der Blockgröße L und erhält dadurch die einzelnen Klartextblöcke $P_1^*, ..., P_s^*$. Da r vom Angreifer mit der Länge $L-1$ Bytes gewählt wurde, gilt nun $P_1^* = r||m[1]$. Anschließend werden alle Klartextblöcke verschlüsselt und das Ergebnis C_0^*,C_1^*,C_1^*,...,C_s^* wird an den Angreifer verschickt. Sei $i = 0$ (Phase 1).
3. Sei der nächste IV der letzte übertragene Block des Orakels, also z.B. C_s^* direkt nach Phase 1. Erzeuge nun $P_i = C_0^* \oplus IV \oplus r||i$ (Phase 2).
4. Erhalte C_i vom Orakel (Phase 2).

5. Wenn $C_i = C_1^*$, gebe i aus, ansonsten erhöhe i und beginne wieder bei Schritt 3 (Phase 2).

Der Algorithmus wird terminieren und nach spätestens 256 Versuchen das richtige Ergebnis für $m[1]$ ermittelt haben, da 1 Byte generell 2^8 verschiedene Werte enthalten kann. Die Vorgehensweise in den Schritten 3-5 (Phase 2) entspricht dabei der bisher gezeigten Vorgehensweise (Abschnitte 3.1-3.3). Es wird versucht das Wissen über den IV auszunutzen, um den Klartext eines Blocks zu erraten. Die Neuerung dabei findet in Phase 1 statt, denn durch die geschickte Wahl von r wird dafür gesorgt, dass nur das letzte Byte unbekannt ist und erraten werden muss anstatt des gesamten Blocks.

Um nun den gesamten Inhalt von m zu ermitteln, erfolgt eine Wiederholung mit einem kürzeren Wert für r. Die Definition von BCBA (Abschnitt 2.2.1.1) stellt dabei sicher, dass es für einen Angreifer möglich ist, mit der richtigen Wahl von r Blockgrenzen zu verschieben. Somit benötigt ein Angreifer für eine Nachricht mit l Bytes durchschnittlich $128 * l$ Versuche, wenn die einzelnen Bytes der Nachricht $(m[1], ..., m[l])$ alle Werte im Bereich $[0 - 255]$ annehmen können. Sollten die Bytes der Nachricht nur einen eingeschränkten Bereich verwenden, sind entsprechend weniger Versuche notwendig. Das würde bedeuten, das Beispiel aus Abschnitt 3.2.2 nochmals aufgegriffen, dass eine 4-stellige PIN nach spätestens $4 * 10 = 40$ Versuchen bekannt ist.

3.4.1 Die manipulierte Webseite

Um diesen Angriff in der Praxis anwenden zu können, müssen vom verwendeten Angriffsvektor die folgenden notwendigen Anforderungen erfüllt werden:

- HTTPS-Anfragen stellen
- Wiederverwendung der SSL-Verbindung bei multiplen HTTPS-Anfragen
- Möglichst freie Kontrolle über die ersten Bytes einer jeden Anfrage

Bei den Autoren sind *HTML5 WebSocket API* (Version 76)[1], *Java URLConnection API* und die *Silverlight WebClient API* als mögliche Angriffsvektoren übrig geblieben.

Der Angreifer würde somit bspw. auf http://mallory.com eine präparierte Webseite bereitstellen, bei der er versucht vom Opfer Zugangsdaten (via HTTP-Header) für https://bob.com zu *erraten*. Browser besitzen zwar Mechanismen wie die Same-Origin-Policy, um http://evil.com an solchen Anfragen zu hindern, aber wie im „Browser Security Handbook“ [Zal11] beschrieben, gibt es nicht die eine Same-Origin-Policy, sondern verschiedene Protokolle interpretieren „same-origin“ unterschiedlich. Zum Beispiel wird von Java „same-origin“ angenommen, wenn http://mallory.com und https://bob.com sich dieselbe IP-Adresse teilen. Beim Einsatz von WebSockets, muss https://bob.com einerseits einen WebSocket zur Verfügung stellen und andererseits Verbindungen von der Webseite des Angreifers zulassen. Das bedeutet umgekehrt, dass der Angreifer in der Lage sein muss, JavaScript auf einer Webseite ausführen zu lassen, die eine WebSocket-Verbindung zu

1 Mit neueren Versionen war der Angriff so nicht mehr möglich, da sie, bedingt durch eine andere bekanntgewordene Sicherheitslücke, entsprechend abgewandelt wurden [Duo11a].

https://bob.com aufbauen darf, was mit Hilfe von Cross-Site Scripting[1] bewerkstelligt werden könnte.

Nicht nur, dass diese Attacke deutlich effizienter als die bislang gezeigte ist, sie benötigt auch keinen Komplizen, der den Netzwerkverkehr des Opfers ausliest.

3.5 2014 - Initialisierungsvektoren sind nicht die einzige Schwachstelle

Im September 2014 haben Bodo Möller, Thai Duong und Krzysztof Kotowicz gezeigt mit „This POODLE bites: exploiting the SSL 3.0 fallback“ [Möl14], dass sich die Idee von BEAST noch in eine andere Art von Angriff verwandeln lässt. Während die bisher gezeigten Attacken die Vorhersagbarkeit des IVs bei SSL ausnutzten, wird bei dem neuen Angriff eine Schwäche des Paddings attackiert. Daher setzt der POODLE (Padding Oracle On Downgraded Legacy Encryption)[2][Möl14] getaufte Angriff eine SSL 3.0 Verbindung voraus, die aber von einem versierten Angreifer leicht erzwungen werden kann.

3.5.1 Aushandeln der SSL-/TLS-Version

Das spezifizierte Verhalten beim Aushandeln der verwendeten SSL-/TLS-Version ist, dass der Client als Teil des Handshakes dem Server seine höchste unterstützte SSL-/TLS-Version mitteilt. Der Server wiederum antwortet mit einer Nachricht, die im Idealfall dieselbe SSL-/TLS-Version beinhaltet. Sollte der Client eine SSL-/TLS-Version genannt haben, die höher ist als die höchste vom Server unterstützte Version, so antwortet der Server mit der von ihm höchsten unterstützten Version. [RFC5246]

Weiter heißt es aber auch im RFC 5246: *Note: some server implementations are known to implement version negotiation incorrectly. For example, there are buggy TLS 1.0 servers that simply close the connection when the client offers a version newer than TLS 1.0. Also, it is known that some servers will refuse the connection if any TLS extensions are included in ClientHello. Interoperability with such buggy servers is a complex topic beyond the scope of this document, and may require multiple connection attempts by the client.* [RFC5246]

Dieser Problematik begegnen Browser mit dem sogenannten „downgrade dance“. Dabei sendet der Browser beim ersten Versuch die höchste von ihm unterstützte SSL-/TLS-Version während des Handshakes mit. Sollte der Handshake scheitern, wird ein neuer Handshake mit einer älteren Version versucht. Dieses Verhalten kann ebenfalls durch einen Angreifer getriggert werden. Eine aktuelle Übersicht über entsprechende Angriffe finden sich in „What's in a Downgrade? A Taxonomy of Downgrade Attacks in the TLS Protocol and Application Protocols Using TLS“ [Ala18]. Die Auswirkungen, die das gezielte Herabsetzen der eingesetzten Protokollversion hat, können als katastrophal bezeichnet werden. Wie in Abbildung 3.3 dargestellt, unterstützten im September 2014 etwa 40% der getesteten 150.000 Webseiten TLS 1.1 und TLS 1.2, während annähernd jede der Webseiten SSL 3.0 unterstützte. Dadurch, dass annähernd jede Webseite, die TLS 1.1 und/oder TLS 1.2

1 Das OWASP führt Cross-Site Scripting Attacken in den Top10 von Bedrohungen für Web-Anwendungen [OWA17].

2 Padding Oracle Attacken bezeichnen Angriffe, bei denen es eine Gegenstelle gibt (das Orakel), die einem Angreifer verrät, ob das Padding eines Ciphertextes korrekt war oder nicht. Die Bezeichnung geht auf „Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS..“ [Vau02] zurück.

unterstützte, ebenso SSL 3.0 unterstützte, konnten durch gezielte Downgrade Attacken die vermeintlich sicheren Verbindungen auf SSL 3.0 herabgesetzt werden.

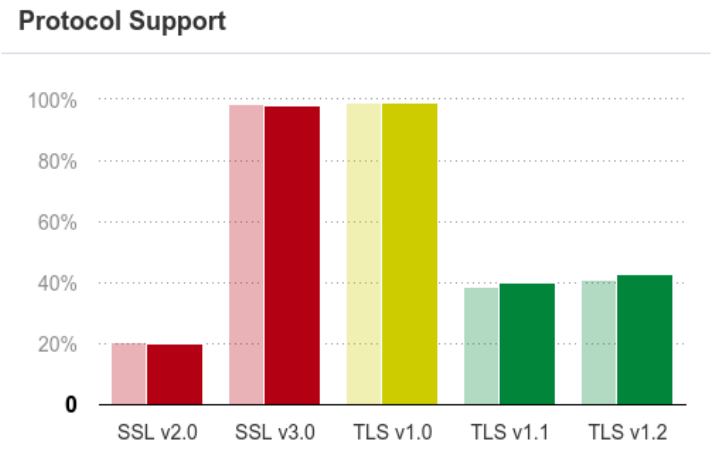

Abbildung 3.3: Verbreitung der SSL-/TLS-Protokolle am 04. September 2014, [Qua19]

3.5.2 Der POODLE-Angriff

Ein großes Problem beim CBC-Mode in SSL 3.0 ist, dass das Padding nicht vom MAC eingeschlossen wird und beliebige Zeichen als Paddingzeichen verwendet werden können. Die Folge daraus ist, dass das Padding während des Entschlüsselns nicht vollkommen verifiziert werden kann.

Wie bereits in Abschnitt 2.1 erläutert, ist es essentiell, dass der gesamte Klartext einem Vielfachen der Blockgröße entspricht. Das bedeutet für SSL 3.0, dass die Länge von Nutzdaten + MAC + Paddingzeichen + Paddinglänge einem Vielfachen der Blockgröße L entspricht.

Die gefundene Schwäche lässt sich am einfachsten ausnutzen, wenn es einen ganzen „Paddingblock“ gibt, welcher vor der Verschlüsselung aus $L - 1$ beliebigen Bytes (Paddingzeichen) gefolgt von einem einzelnen Byte mit dem Wert $L - 1$ (Paddinglänge) besteht.

Bei der Verarbeitung einer ankommenden verschlüsselten Nachricht C, bestehend aus den Blöcken $C_1 \dots C_n$ mit C_0 als IV, ermittelt der Empfänger zuerst $P_1 \dots P_n$, indem er jeweils $P_i = e^{-1}(C_i) \oplus C_{i-1}$ mit $1 <= i <= n$ berechnet. Anschließend werden, soweit notwendig, alle Paddingzeichen entfernt, um am Ende den MAC zu überprüfen und zu entfernen.

Sollte ein „Paddingblock“ C_n existieren und ein Angreifer diesen durch einen früheren Block C_i von derselben Nachricht ersetzen, wird dieser Block ebenfalls akzeptiert, wenn $e^{-1}(C_i) \oplus C_{i-1}$ $L - 1$ als letztes Byte hat, aber ansonsten mit sehr hoher Wahrscheinlichkeit ablehnen. Dies eröffnet die Möglichkeit zu der POODLE getauften Schwäche.

Diese Schwäche von SSL 3.0 kann im Web-Umfeld durch Techniken von BEAST (siehe Abschnitt 3.4) ausgenutzt werden, um HTTP-Cookies auszulesen. Dafür betreibt der Angreifer eine manipulierte Webseite (http://mallory.com), welche den Browser des Opfers dazu veranlasst Anfragen an den Server (https://bob.com) zu senden, von dem der Angreifer

die HTTP-Cookie Informationen des Opfers auslesen möchte. Diese Anfragen muss der Angreifer abfangen und die einzelnen SSL-Blöcke so ändern, dass eine nicht unerhebliche Chance besteht, dass https://bob.com die modifizierten SSL-Nachricht akzeptiert. Wird die geänderte Nachricht wirklich akzeptiert, so kann der Angreifer 1 Byte des Cookies auslesen.

Eine vom Angreifer initiierte Anfrage könnte vor der Verschlüsselung so aussehen:

`POST` */path* `Cookie:` *name=value...*`\r\n\r\n`*body* || 20 Byte MAC || *Paddingzeichen* + *Paddinglänge*

Der Angreifer kann den Inhalt von */path* und *body* bestimmen, was ihm ermöglicht, Anfragen zu verschicken, auf die die beiden folgenden Bedingungen zutreffen:

1. Es wird ein *Paddingblock* C_n verschickt.
2. Das erste unbekannte Byte des Cookies ist das letzte Byte eines früheren Blocks C_i.

Der Angreifer ändert die SSL-Nachricht so ab, dass er C_n durch C_i ersetzt und schickt die manipulierte Nachricht an den Server.

Normalerweise wird der Server diese Nachricht ablehnen. Wenn dies der Fall ist, versucht der Angreifer es erneut. Gelegentlich, durchschnittlich alle 256 Anfragen, akzeptiert der Server eine solche modifizierte Nachricht. Dadurch kann der Angreifer schließen, dass $e^{-1}(C_i)[15] \oplus C_{n-1}[15] = 15$ ist und sich daraus $P_i[15] = 15 \oplus C_{n-1}[15] \oplus C_{i-1}[15]$ ergibt. Somit kennt der Angreifer nun das erste unbekannte Zeichen des Cookies. Nun setzt der Angreifer den Angriff fort, indem er die Größe von *path* und *body* gleichzeitig so verändert, dass die Gesamtgröße der Anfrage gleich bleibt, aber sich die Position des Cookie-Headers so verändert, dass sich an $P_i[15]$ das nächste unbekannte Zeichen des Cookies befindet. Das wiederholt der Angreifer so oft, bis er alle gewünschten Zeichen des Cookies entschlüsselt hat. Der Angreifer muss mit einem Gesamtaufwand von 256 Anfragen pro Byte rechnen.

Literatur

[Ala18] Alashwali, Eman und Kasper Rasmussen: „What's in a Downgrade? A Taxonomy of Downgrade Attacks in the TLS Protocol and Application Protocols Using TLS“. 2018 (siehe S. 15).

[And08] Anderson, Ross: *Security Engineering: A Guide to Building Dependable Distributed Systems, Second Edition*. Wiley Publishing, Inc., 2008 (siehe S. 3).

[Arb15] Arbeitsgruppe für Kryptographie und Sicherheit, Institut für Theoretische Informatik: *Sicherheit - Skript zur Stammvorlesung*. Karlsruher Institut für Technologie, 2015 (siehe S. iii).

[Bar06] Bard, Gregory V: „A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL.“ 2006: S. 99–109 (siehe S. 11).

[Bar04] Bard, Gregory V: „The Vulnerability of SSL to Chosen Plaintext Attack.“ (2004), Bd. 2004(111) (siehe S. 4, 5, 8, 9).

[RFC7568] Barnes, R., M. Thomson, A. Pironti und A. Langley: *Deprecating Secure Sockets Layer Version 3.0*. RFC 7568. `http://www.rfc-editor.org/rfc/rfc7568.txt`. RFC Editor, 2015 (siehe S. 4).

[Dai02] Dai, Wei u. a.: *An attack against SSH2 protocol*. 2002. URL: `https://ietf-ssh.netbsd.narkive.com/UjgKo2TK/an-attack-against-ssh2-protocol` (siehe S. 7, 8).

[RFC4346] Dierks, T. und E. Rescorla: *The Transport Layer Security (TLS) Protocol Version 1.1*. RFC 4346. `http://www.rfc-editor.org/rfc/rfc4346.txt`. RFC Editor, 2006 (siehe S. 4).

[RFC5246] Dierks, T. und E. Rescorla: *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. `http://www.rfc-editor.org/rfc/rfc5246.txt`. RFC Editor, 2008 (siehe S. 4, 15).

[RFC2246] Dierks, Tim und Christopher Allen: *The TLS Protocol Version 1.0*. RFC 2246. Published: Internet Requests for Comments. RFC Editor, 1999 (siehe S. 4, 5).

[Duo11a] Duong, Thai: *BEAST*. 2011. URL: `https://vnhacker.blogspot.com/2011/09/beast.html` (siehe S. 14).

[Duo11b] Duong, Thai und Juliano Rizzo: „Here come the ⊕ ninjas“. (2011), Bd. 320 (siehe S. 3, 4, 12, 13).

[RFC2616] Fielding, Roy T., James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach und Tim Berners-Lee: *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. `http://www.rfc-editor.org/rfc/rfc2616.txt`. RFC Editor, Juni 1999 (siehe S. 5).

[RFC6101] Freier, A., P. Karlton und P. Kocher: *The Secure Sockets Layer (SSL) Protocol Version 3.0*. RFC 6101. Published: Internet Requests for Comments. RFC Editor, 2011 (siehe S. 4, 5).

[ITU94] ITU: *Open Systems Interconnection - Model and Notation*. 1994. URL: `http://handle.itu.int/11.1002/1000/2820` (siehe S. 4, 5).

[Jou02] Joux, Antoine, Gwenaëlle Martinet und Frédéric Valette: *Blockwise-Adaptive Attackers: Revisiting the (In)Security of Some Provable Secure Encryption Modes: CBC, GEM, IACBC*. 2002 (siehe S. 3).

[Kra01] Krawczyk, Hugo: *The order of encryption and authentication for protecting communications (Or: how secure is SSL?)* Published: Cryptology ePrint Archive, Report 2001/045. 2001 (siehe S. 6).

[Men96] Menezes, Alfred J., Paul C. van Oorshot und Scott A. Vanstone: *Handbook of Applied Cryptography*. CRC Press, 1996 (siehe S. iii).

[Möl04] Möller, Bodo: *Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures*. 2004. URL: `https://www.openssl.org/~bodo/tls-cbc.txt` (siehe S. 12).

[Möl14] Möller, Bodo, Thai Duong und Krzysztof Kotowicz: „This POODLE bites: exploiting the SSL 3.0 fallback“. *Security Advisory* (2014), Bd. (siehe S. 15).

[Mor19] Moriarty, Kathleen und Stephen Farrell: *Deprecating TLSv1.0 and TLSv1.1*. Internet-Draft draft-ietf-tls-oldversions-deprecate-05. Work in Progress. Internet Engineering Task Force, 2019 (siehe S. 4).

[OWA17] OWASP: „OWASP Top 10 2017“. (2017), Bd. (siehe S. 15).

[Paa10] Paar, Christof und Jan Pelzl: *Understanding Cryptography: A Textbook for Students and Practitioners*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010 (siehe S. 2).

[Qua19] Qualys Inc.: *SSL Pulse*. 2019. URL: `https://www.ssllabs.com/ssl-pulse/` (siehe S. 16).

[RFC8446] Rescorla, E.: *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. RFC Editor, 2018 (siehe S. 4).

[Rog95] Rogaway, Phil: *Problems with Proposed IP Cryptography*. 1995. URL: `https://web.cs.ucdavis.edu/~rogaway/papers/draft-rogaway-ipsec-comments-00.txt` (siehe S. 7).

[Sar14] Sarukkai, Sekhar: *POODLE – How bad is its bite? (Here's the data)*. 2014. URL: `https://www.skyhighnetworks.com/cloud-security-blog/poodle-how-bad-is-its-bite-here-is-the-data/` (siehe S. 1).

[RFC6176] Turner, S. und T. Polk: *Prohibiting Secure Sockets Layer (SSL) Version 2.0*. RFC 6176. `http://www.rfc-editor.org/rfc/rfc6176.txt`. RFC Editor, 2011 (siehe S. 4).

[Vau02] Vaudenay, Serge: „Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS...“ *Advances in Cryptology — EUROCRYPT 2002*. Springer Berlin Heidelberg, 2002 (siehe S. 15).

[Zal11] Zalewski, Michal: *Browser Security Handbook*. 2011. URL: `https://code.google.com/archive/p/browsersec/wikis/Main.wiki` (siehe S. 14).

Abbildungsverzeichnis