Client-Server-Kommunikation im Detail

IB

Benjamin Mayer

1823720

Inhaltsverzeichnis

1 Netzwerkadressierung

In diesem Kapitel werden Sie die grundlegenden Technologien und Standards kennenlernen, die Kommunikation zwischen verschiedenen Computern ermöglichen und wie man die Nutzung dieser Technologien menschenfreundlich gemacht hat.

Nach diesem Kapitel sollten Sie

- *wissen wie sich Computer gegenseitig adressieren und identifizieren*
- *die einzelnen Teile einer Internetadresse erkennen und nennen können.*

1.1 IP-Adresse

Die IP-Adresse (Internet Protocol Adresse) eines Computers ist analog zur Adresse einer Person zu verstehen. Soll ein Brief an eine bestimmte Person versendet werden, muss man zuerst herausfinden, unter welcher Adresse man diese Person findet. Will man einem Computer eine Nachricht schicken braucht man dessen IP-Adresse. Die IP-Adresse ist somit eine der notwendigen Komponenten, um die Kommunikation zwischen Computern zu ermöglichen. Heutzutage sind hauptsächlich zwei Versionen der IP-Adressen in Nutzung, IPv4 und IPv6.

1.1.1 IPv4

IPv4 (Internet Protocol Version 4) ist die gängigste Version des Internet Protocols. Eine IPv4-Adresse besteht aus vier Zahlen zwischen 0 und 255. Dargestellt wird eine solche Adresse durch die vier Zahlen, getrennt durch Punkte, wie in Abb. 1.1.1 zu sehen.

```
206.34.0.168
```

[Abb. 1.1.1] Eine IPv4-Adresse

Durch Kombination der vier Zahlen ergibt sich eine Adressraum von über 4 Milliarden Adressen. Trotz der großen Anzahl von Adressen, sind mittlerweile IPv4-Adressen in Amerika schon knapp (vgl. [HIP4A]) und in Europa kann die Institution zur Adressverwaltung RIPE (Réseaux IP Européens) überhaupt keine neuen IPv4-Adressen ausgeben (vgl. [HIP4E]).

1.1.2 IPv6

IPv6 (Internet Protocol Version 6) ist eine neuere Version des Internet Protocols. Die größte Änderung von IPv4 ist die Größe der Adresse von 32 Bit auf 128 Bit, wodurch der Adressraum von knapp 4,3 Milliarden auf 340 Sextillionen erhöht wird. Damit wird auf das Problem reagiert, dass man mit 32 Bit nicht genug verschiedene IP-Adressen bereitstellen kann, um der Nachfrage gerecht zu werden.

Wegen ihrer Größe wird die IPv6-Adresse in Hexadezimalen Ziffern dargestellt. Die Adresse besteht aus acht Blöcken mit je einer hexadezimalen Zahl, welche Werte zwischen 0000 und ffff (dezimal: 0-65535) annehmen können. Wie in Abb. 1.1.2 zu sehen ist, dürfen führende Nullen weggelassen und Blöcke, die nur aus Nullen bestehen, komplett ausgelassen werden. Das darf jedoch nur einmal in der Adresse gemacht werden, um die Adresse jederzeit eindeutig rekonstruieren zu können. Außerdem dürfen führende Nullen eines Blocks weggelassen werden.

```
      03c4:8a35:0003:0000:0000:003d:0000:0000
<=>   3c4:8a35:3:39f1::3d:0:0
<=>   3c4:8a35:3:39f1:0:0:3d::
```

[Abb. 1.1.2] Dieselbe IPv6-Adresse unterschiedlich gekürzt

1.2 Sockets und Ports

TCP (Transmission Control Protocol) -Ports ermöglichen, in Kombination mit der IP-Adresse, die Kommunikation zwischen Client und Server. Mithilfe des Ports (dt. Hafen) wird die genaue Schnittstelle angegeben, an der die Kommunikation stattfinden soll. Das ermöglicht zwei verschiedene Verbindungen zwischen denselben zwei Sockets zu unterscheiden.

Definition 1.1: Socket

Eine Socket (dt. Buchse) ist ein Endpunkt bei der Client-Server-Kommunikation. Ein Endpunkt ist dabei immer ein Computer.

Eine Portnummer kann überall zwischen 1 und 65535 sein. Diese sind jedoch aufgeteilt wie folgt:

- 1 – 1023 System bzw. Well-Known Ports
- 1024 – 49151 User bzw. Registered Ports
- 49152 – 65535 Dynamic bzw. Private oder Ephemeral Ports

Vgl. [RFCTCP]

Ein Server öffnet in der Regel Port 80, einer der Well-Known Ports, um Anfragen von Clients anzunehmen. An diesem Port wartet der Server dauerhaft, bis Anfragen vom Client ankommen.

Der Client öffnet meist einen zufälligen Port, da er diesen dem Server sowieso mitteilt, sobald eine Anfrage vom Client an den Server gestellt wird. Will der Client verschiedene Verbindungen mit demselben Server parallel aufbauen, beispielsweise um verschiedene Dateien gleichzeitig herunterzuladen, kann er neue Ports öffnen, auf denen er weitere Anfragen an denselben Port 80 des Servers schickt.

1.3 URI

Definition 1.2: URI

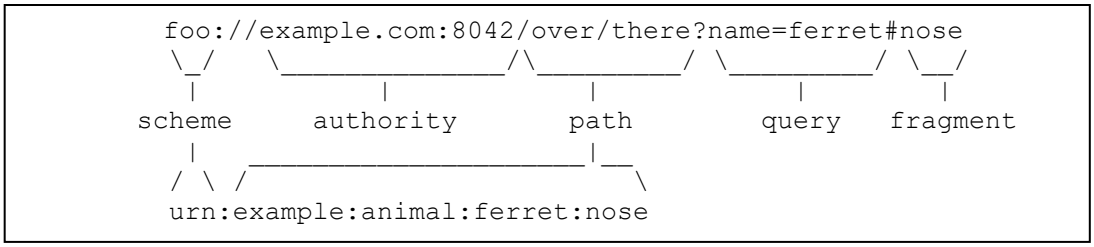
URI bedeutet Uniform Resource Identifier (dt. einheitlicher Bezeichner für Ressourcen). URIs dienen dazu jegliche Ressourcen wie Webseiten, Dateien oder Webservices im Internet zu identifizieren.

1.3.1 Syntax

Eine URI beginnt immer mit dem Scheme (dt. Schema) worauf immer ein „:“ folgt. Gibt es eine Authority (dt. ein Zuständiger) wird dieser mit „//“ eingeleitet. In der Authority steht entweder direkt eine IP-Adresse und Port des Zuständigen (Server) oder ein Domain Name, der von einem DNS-Server in eine IP-Adresse und Port übersetzt wird.

Definition 1.3: DNS-Server

Auf DNS (Domain Name Service) -Servern laufen Anwendungen, welche vom Menschen einfach lesbar und einprägsame Domain Namen (z.B. www.hs-mannheim.de) zusammen mit der dazugehörigen IP-Adresse speichern. Wenn diese Domain angefragt wird wandelt der DNS-Server den menschenlesbaren Domain Namen in eine IP-Adresse um.



[Abb. 1.3.1a] Syntax-Beispiel einer URI

Der Path (dt. Pfad) nach einer Authority gibt den Pfad zu der Ressource an, die vom Server zurückgeschickt werden soll. Der Path kann ebenso wie die Authority leer sein. Ist die Authority leer, wird nach Schema und „:“ nicht mit „//“ eingeleitet. Ist die Authority nicht leer, muss der Path entweder leer sein oder mit „/“ beginnen. Beendet wird der Path durch ein „?“ einer Query, ein „#“ eines Fragments oder dem Ende der URI.

Eine Query (dt. Anfrage) liefert dem Server Daten in Form von Key-Value-Paaren mit. Eingeleitet wird die Query mit „?“ und Key-Value-Paare werden getrennt durch ein „&“-Zeichen, wie in Abb. 1.3.1b zu sehen. Beendet wird die Query mit dem Anfang eines Fragments durch „#“ oder dem Ende der URI.

```
?user=horst&passwort=1234
```

[Abb. 1.3.1b] Eine Query mit zwei Key-Value-Paaren

Ein Fragment zeigt auf ein bestimmten Element der aufgerufenen Ressource. Somit kann ein Webbrowser beispielsweise direkt an eine bestimmte Stelle einer Webseite springen und erspart dem User das Suchen bzw. Scrollen.

1.3.2 URL & URN

URL und URN sind Unterkategorien von URI.

Ein Uniform Resource Locator (URL, dt. einheitlicher Ressourcenzeiger) ist die am häufigsten verwendete Art von URIs. Sie zeigen, wo die gewollte Ressource, meist Webseiten, auf dem Server zu finden ist.

Die Syntax einer URL folgt denselben Regeln wie die der URI. Im Normalfall ist das genutzte Schema entweder HTTP oder HTTPS, die beiden Protokolle, um Webseiten im Internet zu verschicken. In diesem Fall folgt dem Schema der Domain-Name, wie beispielsweise „www.wikipedia.org“, worauf wiederum ein Pfad, dann eine Query und dann ein Fragment folgen kann aber nicht muss. Ein gesamtes Beispiel wäre dann „https://de.wikipedia.org/wiki/Uniform_Resource_Locator#Aufbau“, wobei nur eine Query fehlt.

Mit einem Uniform Resource Name (URN, dt. einheitlicher Ressourcenname) wird eine bestimmte Ressource, wie beispielsweise wissenschaftliche Werke, eindeutig und dauerhaft identifiziert. Zu URNs gelten alle URIs mit dem Schema „urn“. Mit einem Doppelpunkt getrennt folgt auf das Schema ein Namespace Identifier (NID), der wiederum mit einem Doppelpunkt abgeschlossen wird. Was darauf folgt wird vom jeweiligen Namespace (dt. Namesraum) festgelegt. Einer dieser Namensräume ist der der Internationalen Standardbuchnummer (ISBN). In diesem Namensraum würde eine URI wie folgt aussehen: „urn:ISBN:<ISBN des Buches>“. Dies ist jedoch nur einer von vielen möglichen Namensräumen.

Zusammenfassung

In diesem Kapitel wurden die Einzelteile der Netzwerkadresse vorgestellt, deren Nutzen erklärt sowie deren Vor- und Nachteile besprochen. Daraufhin wurde der allgemeine Standard zur Identifikation von Ressourcen im Internet (URI) definiert und anhand von Beispielen erklärt. Letztlich wurde die Syntax der beiden Standards URL und URN erklärt.

Aufgaben zur Selbstüberprüfung

Aufgabe 1.1

Nennen Sie die zwei Versionen der IP-Adresse, welche aktuell in Verwendung sind und geben Sie ein Beispiel für beide.

Aufgabe 1.2

An welchem Port lauscht ein Server standardmäßig auf Anfragen?

Aufgabe 1.3

Nennen Sie die fünf Teile einer URI.

Aufgabe 1.4

Es sollen die beiden Attribute „username“ und „pin“ per URI an den Server übertragen werden. Welcher Teil der URI ist dafür zuständig und wie würde er mit korrekter Syntax aussehen, wenn „username“ den Wert „hschneider“ und „pin“ den Wert „1234“ annehmen soll?

2 Datenstrukturen

In diesem Kapitel lernen Sie die verschiedenen Sprachen und Metasprachen kennen, die verwendet werden, um Datenstrukturen fest zu definieren. Jede dieser Sprachen spielt eine wichtige Rolle in der Web-Kommunikation oder deren Entwicklung.

Nach diesem Kapitel sollten sie wissen

- *für welchen Anwendungszweck die jeweiligen Sprachen gedacht sind*
- *welche Vor- und Nachteile jede Sprache mit sich bringt*
- *wie sich die Sprachen entwickelt haben.*

2.1 XML

Die Extensible Markup Language (XML) entstand als Weiterentwicklung von SGML (Standard Generalized Markup Language), einer Meta-Sprache, welche entwickelt wurde, um neue Sprachen zu definieren. XML ist somit also auch eine Metasprache. SGML hat jedoch viele Freiheiten, welche die Sprache für Menschen womöglich verständlicher machen, die es aber schwierig machten, die Sprache vom Computer zu verarbeiten und zu analysieren. Um also eine für Computer besser lesbare Sprache zu entwickeln, wurden strengere Syntaxregeln eingeführt.

Jedes XML-Dokument hat denselben Aufbau. Es besteht aus mehreren ineinander geschachtelten Elementen, deren Anfang und Ende mit Tags gekennzeichnet sind. Diese sehen wie folgt aus: Anfang: „<example>“, Ende: „</example>“. Befindet sich zwischen Anfang- und Endtag kein Inhalt, kann man beide auch zusammenführen: „<example/>“. Sind verschiedene Elemente ineinander verschachtelt, muss ein Element immer geschlossen werden, bevor das Elternelement mit einem Endtag geschlossen wird oder ein Nachbarelement mit einem Anfangstag geöffnet wird.

2.1.1 DOM-Bäume

Weil Elemente in XML so streng strukturiert geschachtelt sind, kann man sie mithilfe eines DOM (Document Object Model) -Baumes darstellen. Dies ist möglich mit jeder Art von Datei, die durch XML oder SGML definiert wurde.

Beginnend mit dem Wurzelelement ist jedes Element in der XML-Datei ein Knoten im Baum. Jedes Element, welches in ein anderes Element geschachtelt ist, ist dessen Kind. Veranschaulicht wird dieses Konzept in Abb. 2.1.1, anhand einer XML-Datei, die einen Studenten beschreibt. Im zugehörigen DOM-Baum sieht man, dass das XML-Dokument selbst die Wurzel des Baumes ist. Darin befindet sich das student-Objekt, worin sich wiederum die drei Eigenschaften des Studenten befinden.

[Abb. 2.1.1] XML Datei mit zugehörigem DOM-Baum

2.1.2 DTD

Eine Datentypdefinition (DTD) definiert die Grammatik, an die sich ein XML halten muss, damit das entstandene Dokument dem gewünschten Datentyp entspricht. In der DTD wird angegeben, welche Elemente der Datentyp hat und auf welche Art diese Elemente geschachtelt werden.

DTDs haben drei verschiedene Typen, die definiert werden können: Elemente, Attribute und Entitäten. Elemente werden durch ELEMENT eingeleitet und können beliebigen Inhalt haben. Entweder sie enthalten weitere Elemente oder sie enthalten #PCDATA (Zeicheninhalte), EMPTY (kein Inhalt) oder ANY (beliebigen Inhalt). Auch Anzahl und Reihenfolge können angegeben werden. Enthält ein Element mehr als ein weiteres Element, können diese mit Klammern gruppiert werden. Die Reihenfolge innerhalb der Klammer wird durch eine mit Kommata getrennte Auflistung festgelegt. Soll es Alternativen geben werden diese jeweils durch „|“ getrennt. Um anzugeben, wie oft ein bestimmtes Element vorhanden ist, werden „*“ für beliebig oft, „+“ für mindestens ein Mal und „?“ für genau ein oder kein Mal, verwendet. Ist keines dieser Zeichen vorhanden, muss das Element genau ein Mal vorhanden sein. Abb. 2.1.2 zeigt eine DTD zum in Abb. 2.1.1 vorgestellten Beispiel, um die Syntax klar zu machen.

Da jedoch die Syntax der DTDs an reguläre Ausdrücke erinnert und somit nicht leicht lesbar für den Computer war, wurde eine Alternative entwickelt.

```
<!DOCTYPE student
[
  <!ELEMENT student (vorname, nachname, matrikelnummer)>
  <!ELEMENT vorname (#PCDATA)>
  <!ELEMENT nachname (#PCDATA)>
  <!ELEMENT matrikelnummer (#PCDATA)>
]>
```

[Abb. 2.1.2] DTD zur XML-Anwendung student

2.1.3 XML-Schemata

XML-Schemata haben dieselbe Funktion wie DTDs, jedoch mit einigen neuen Möglichkeiten. Anstatt die Grammatik mithilfe regulärer Ausdrücke zu beschreiben wird sie in Form eines XML-Dokumentes definiert. So haben alle Elemente klare Anfangs- und Endtags, die sich leichter vom Computer lesen lassen.

Mit XML-Schemata kann man zwischen simplen und komplexen Datentypen unterscheiden. Ein simpler Typ nimmt hierbei nur einen der von XML-Schemata gegebenen atomare Datentypen an, während einige Einschränkungen und Grenzwerte definiert werden. Komplexe Datentypen wiederum beinhalten nicht nur einen atomaren Datentyp, sondern einen bis viele Elemente, die selbst von einem atomaren oder auch benutzerdefinierten Datentyp sein können.

XML-Schemata bietet die atomaren Datentypen wie man sie aus anderen Programmiersprachen kennt; String, Decimal, Integer, Float, Boolean, Date, Time und auch einige neue, speziell für XML-Schemata. Durch diese Änderungen lassen sich Datentypen genauer definieren während sie gleichzeitig für Computer einfacher lesbar sind.

Im Kontrast zu Abb. 2.1.2 zeigt Abb. 2.1.3 ein XML-Schema zu Abb.2.1.1

```
<xs:complexType name="student">
    <xs:sequence>
        <xs:element name="vorname"          type="xs:string"/>
        <xs:element name="nachname"         type="xs:string"/>
        <xs:element name="matrikelnummer"   type="xs:integer"/>
    </xs:sequence>
</xs:complexType>
```

[Abb. 2.1.3] XML-Schema zum Typ student

2.2 HTML

Die Hypertext Markup Language (HTML) ist eine rein textbasierte Sprache, die zur Strukturierung von online gestellten Dokumenten entwickelt wurde. Definiert wurde sie durch SGML, wodurch sie ähnliche Makel hat. Die Syntax von HTML-Dokumenten ist nicht besonders streng, was es demjenigen, der es schreibt, zwar einfacher macht, da er nicht auf so viele Dinge achten muss. Jedoch hat das zur Folge, dass die Sprache schwieriger von Computern lesbar ist.

Der generelle Aufbau einer HTML-Datei besteht aus Anfang- und Endtags verschiedener Elemente. Die Tags haben denselben Aufbau wie die eines XML-Dokuments. Die Hauptelemente des HTML-Datentyps sind `<html>`, `<head>` und `<body>`. Das `<html>`-Element ist das Wurzelelement des Dokuments. In ihm befinden sich `<head>` und `<body>`.

Im Head kann mit `<title>` ein Titel angegeben werden oder mit `<link>`, `<script>` oder `<object>` jeweils Beziehungen zu anderen Ressourcen, Code einer Skriptsprache oder externe Dateien angeben bzw. einbinden. Man kann außerdem mit `<meta>` verschiedene Metadaten des Dokuments angeben und mit `<style>` die Darstellung des Dokuments im Browser verändern. Wobei letzteres mittlerweile zwar noch möglich ist wird es jedoch meist in ein Cascading Style Sheet (CSS) ausgelagert, welches durch `<link>` wieder eingebunden wird.

Der Body beinhaltet den eigentlichen Inhalt des Dokuments, den Text, der auf der Webseite angezeigt wird. Dieser kann in verschiedenen Elementen aufgeteilt werden, die das letztendlich Aussehen entscheiden. Beispielsweise gibt es verschieden große Überschriften mit `<h1>` bis `<h6>`, die immer eine Ebene tiefer gehen und mit `<p>` bekommt man einen simplen Paragrafen. Einige Elemente haben auch Attribute, die man im Tag in Form eines Key-Value-Paars mit der Syntax „key=“value“ “ angibt.

Die Syntax von HTML lässt dem Entwickler jedoch viele Freiheiten. Es ist beispielsweise möglich das Anfang- oder Endtag eines Elements wegzulassen. Bei Elementnamen muss auch nicht auf Groß- und Kleinschreibung geachtet werden.

2.3 XHTML

Nachdem HTML als Anwendung von SGML entstanden ist und SGML mittlerweile von XML mit seiner besseren Syntax überholt wurde, hat man sich überlegt auch HTML auf Basis von XML neu zu definieren. Was dabei entstand ist heute als XHTML (Extenstible HTML) bekannt.

Da XHTML komplett auf XML-Basis definiert ist, ist zwar die Syntax strenger als bei HTML. Das bedeutet wiederum aber auch, dass XHTML sich, genau wie alle anderen durch XML definierte Sprachen, einfach von Programmen auslesen und verarbeiten lässt.

Neben vielen kleinen Änderungen sind die wichtigsten Änderungen, die XHTML vornimmt sind Folgende. Alle Tags, die geöffnet werden, müssen auch explizit wieder geschlossen werden. Alle Elemente und Attribute müssen in ausschließlich in Kleinbuchstaben angegeben werden. Die Elemente `<html>`, `<head>`, `<title>` und `<body>` müssen immer angegeben werden.

Durch die Änderungen in XHTML wird die Syntax zwar verschärft, aber die grundsätzlichen Elemente, Attribute und Verschachtelungsregeln, die in HTML vorhanden waren, sind auch in XHTML vorhanden. In Kombination mit den anderen Vorteilen der Sprache, war dies der Grund dafür, dass der XHTML-Standard in den HTML5-Standard, der neusten Version von HTML, übernommen wurde.

2.4 JSON

JavaScript Object Notation (JSON) ist neben XML eine weitere Art Datenstrukturen in reinem Text festzuhalten. Es wird hauptsächlich genutzt, um Daten abzuspeichern oder sie zwischen verschiedenen Anwendungen auszutauschen. Jedes JSON-Dokument ist gültiger JavaScript Code, JSON kann jedoch auch durch Einlesen und Interpretieren des reinen Textes in anderen Sprachen ausgelesen werden.

Auch JSON hat die typischen atomaren Datentypen, die es von JavaScript erbt. Dazu gehören der Nullwert „null“, boolesche Werte „true“ und „false“ sowie Zahlen, Zeichenketten Arrays und auch komplexere Objekte. Objekte werden durch „{“ eingeleitet und enden mit „}“. Sie können Eigenschaften haben, die nach der Syntax „Schlüssel: Wert“ angegeben werden. Schlüssel müssen immer Zeichenketten sein und sollten eindeutig sein. Werte können einer von JSONs Datentypen sein. Abb. 2.4 zeigt die JSON-Syntax anhand desselben Beispiels wie Abb. 2.1.1, um den Vergleich zwischen JSON und XML zu bieten.

```
{
    "student": {
        "vorname":          "Hans",
        "nachname":         "Müller",
        "matrikelnummer":   "1823340"
    }
}
```

[Abb. 2.4] JSON zum Object student

2.5 HTML vs. XML vs. JSON

Nachdem nun verschiedene Konzepte der Datenstrukturierung vorgestellt wurden, gilt es deren Vor- und Nachteilen in bestimmten Aspekten zu zeigen.

HTML ist die Sprache der drei, die am wenigsten flexibel ist, da sie ausschließlich zur Strukturierung von Webseiten gedacht ist. XML und JSON hingegen sind nicht auf einen Datentyp limitiert. Beide Sprachen sind flexibel. Jedoch gibt es auch hier Unterschiede.

Während beide in der Lage sind beliebige Datenstrukturen zu speichern, hat nur XML die Möglichkeit anhand eines XML-Schemas oder einer DTD die Datenstruktur festzulegen. Somit kann an einer Schnittstelle fest definiert werden, welche Datentypen übertragen werden. Es kann jedoch auch von Vorteil sein diese Einschränkung nicht zu haben. Wenn man also eine flexible Schnittstelle braucht ist JSON besser geeignet.

Ein weiterer Vorteil von JSON ist, dass jede JSON-Datei per JavaScript mit der Methode eval() evaluiert werden kann. Das bedeutet alle in der JSON-Datei gespeicherten Objekte werden in JavaScript-Objekte umgewandelt. Diese Funktionalität sollte jedoch mit Vorsicht verwendet werden, da so auch schädlicher Code ohne Weiteres ausgeführt wird.

Zusammenfassung

In diesem Kapitel wurden zunächst die Metasprachen XML und SGML und deren Funnktion vorgestellt. Daraufhin wird beschrieben, wie HTML und XHTML mithilfe von SGML bzw. XML entstanden ist. Zuletzt wird JSON vorgestellt und mit HTML und XML verglichen.

Aufgaben zur Selbstüberprüfung

Aufgabe 2.1

Gegeben ist folgendes XML-Dokument:

```xml
<?xml version="1.0"?>
<produkt>
  <name>Sessel</name>
  <stueckzahl>243</stueckzahl>
  <preis>99.99</preis>
  <lagerplatz>A5</lagerplatz>
</produkt>
```

Zeichnen Sie den dazugehörigen DOM-Baum.

Aufgabe 2.2a

Geben Sie eine DTD an, die den Datentyp im XML-Dokument von Aufgabe 2.1 definiert.

Aufgabe 2.2b

Geben Sie ein XML-Schema an, das den Datentyp im XML-Dokument von Aufgabe 2.1 definiert.

Aufgabe 2.3

Wandeln Sie das XML-Dokument aus Aufgabe 2.1 in ein JSON-Dokument um.

3 Client-Server-Kommunikation

Dieses Kapitel definiert notwendige Begriffe und zeigt den grundsätzlichen Aufbau der Kommunikation zwischen Client und Server.

Sie sollten nach diesem Kapitel wissen

- *wie HTTP Daten überträgt und welche Risiken dies mit sich bringt*
- *wie HTTPS die Nachteile von HTTP ausbessert*
- *wie Datenübertragungen ablaufen und wie sie strukturiert sind*
- *welche Möglichkeiten der Client hat, mit dem Server zu interagieren*

3.1 HTTP

Das Hypertext Transfer Protocol (HTTP) ist ein zustandsloses Protokoll zur Datenübertragung und wird meistens dazu genutzt, Webseiten aus dem Internet in einen Browser zu laden. Um Persistenz in HTTP zu schaffen, muss der Server jedem Client eine Session-ID zuordnen und diese speichern.

HTTP ist ein reines Klartextprotokoll, was bedeutet, dass alle Daten, die per HTTP übertragen werden, auf keine Weise verschlüsselt sind und jeder, der sich im selben Netzwerk befindet, diese einfach mitlesen kann. Daher bietet HTTP keine Sicherheit der Daten.

3.1.1 HTTPS

HTTPS (Hypertext Transfer Protocol Secure) versucht dieses Problem der Sicherheit bei der Datenübertragung zu lösen. Es ist umgesetzt als zusätzliche Schicht zwischen HTTP und TCP. Dort werden die Daten verschlüsselt und eine Authentifizierung findet statt. Dadurch wissen Server und Client wer am anderen Ende der Verbindung ist.

Die Verschlüsselung der Daten wird mithilfe von Transport Layer Security (TLS, ehemals SSL bzw. Secure Sockets Layer) umgesetzt. TLS besteht selbst aus zwei Hauptkomponenten, das TLS Record Protocol und das TLS Handshake Protocol.

Das Handshake Protocol authentifiziert die beiden Teilnehmern der Verbindung, ermittelt, welche Kryptographie-Methoden und Parameter genutzt werden, und erstellt die Schlüssel.

Das Record Protocol nutzt die gegebenen Parameter des Handshake Protocols, um den Datenfluss der beiden teilnehmenden Parteien zu schützen. Vgl. [RFCTLS]

Da die Verbindung selbst nach diesen Vorkehrungen noch anfällig für Man-in-the-Middle Angriffe ist, muss sich der Server durch ein X.509-Zertifikat authentifizieren. Diese Zertifikate werden von offiziellen Zertifizierungsstellen erteilt. Die meisten Browser erkennen anerkannte Zertifizierungsstellen und vertrauen deren Zertifikate.

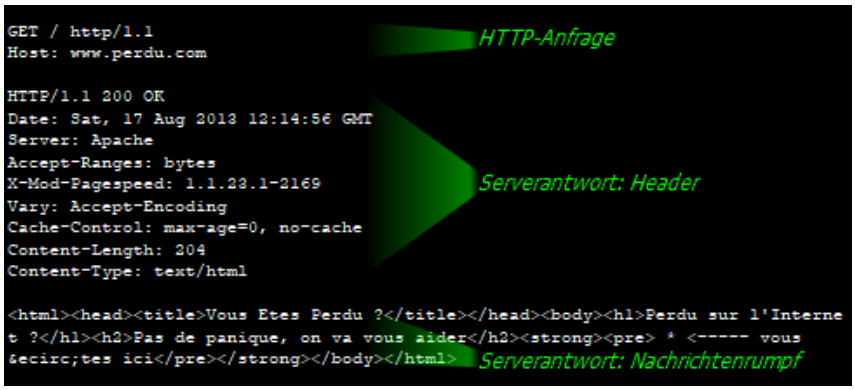
Definition 2.1: Man-in-the-Middle Angriff

Ein Angriff, bei dem sich ein Dritter unbemerkt in die Kommunikation einschaltet, sich gegenüber A für B und gegenüber B für A ausgibt – und alle übertragenen Daten mitlesen kann. Vgl. [ITSM]

3.1.2 HTTP-Nachrichten

Eine HTTP-Nachricht besteht aus zwei Teilen. Dem Header (Kopf) und dem Body (Körper/Rumpf).

Der Header enthält mehrere Felder mit Metadaten, wie beispielsweise dem Datentyp, der im Body gesendet wird. Im Beispiel der Abb. 3.1.2 wird eine Textdatei vom Typ HTML übertragen.



[Abb. 3.1.2] Ein HTTP-Request und Response mit Header und Body

Der Body einer HTML-Nachricht sind die eigentlich zu übertragengenden Daten, in diesem Fall das HTML-Dokument.

3.1.2.1 Wichtige Header-Felder

Es gibt insgesamt fast 50 mögliche Header-Felder für eine HTTP-Nachricht, wovon viele jedoch nicht häufig genutzt werden. Zu den häufig verwendeten gehören:

- Date – Zeit des Requests oder der Response
- Content-Type – Der Datentyp nach dem gefragt wird bzw. der geschickt wird
- Content-Length – Die Größe des Bodys der Nachricht in Byte
- Content-Encoding – Wie die Datei kodiert wurde
- Last-Modified – Wann die Daten zuletzt geändert wurden

Speziell für Request-Nachrichten:

- Host – Der Domain-Name des Servers
- If-Modified-Since – Server schickt nur die angefragte Datei, falls diese seit dem angegebenen Datum verändert wurde
- Accept-Language – Die Sprachen, die der Client kennt
- User-Agent – Informationen über den Browser des Users

Speziell für Response-Nachrichten:

- Server – Informationen über den Server
- WWW-Authenticate – Erzwingt eine Authentifizierung

Vgl. [RFCHF]

3.2 MIME-Typen

MIME (Multi Purpose Internet Mail Extensions) ist ein Standard, um Datentypen anzugeben. Sie werden genutzt, um die Datentypen in einer HTTP-Nachricht zu spezifizieren.

Die generelle Syntax eines MIME-Typen ist „Medientyp/Subtyp“. Wird eine HTML-Datei versendet ist deren MIME-Typ beispielsweise „text/html“. Bei einem JPEG-Bild wäre es „image/jpeg“. Häufig verwendete Dateien und deren MIME-Typen sind in Tab. 3.2 zu sehen.

Dateityp	MIME-Typ
.html	text/html
.css	text/css
.xml	text/xml
.mp4	audio/mp4
.jpeg	image/jpeg
.mpeg	video/mpeg
.pdf	application/pdf

[Tab. 3.2] Häufig verwendete MIME-Typen

3.3 Request & Response

Die gesamte Client-Server-Kommunikation beruht auf dem Request-Response-Prinzip. Der Client gibt eine URI ein, im Normalfall eine URL, die von einem DNS-Server zur IP-Adresse des Servers umgewandelt wird. Daraufhin wird am für HTTP standardisierten Port 80 des Servers angefragt. Bei der Anfrage schickt der Client seine IP-Adresse und Port mit und lässt den Server so wissen, wohin er die Antwort schicken soll. Da das HTTP-Protokoll zustandslos ist, merkt sich weder Client noch Server mit wem er kommuniziert hat. Deswegen muss bei jeder neuen Nachricht die Adresse mitgeschickt werden.

3.3.1 Request-Nachricht

Eine Request-Nachricht enthält den Namen der Request-Methode, die vollständige URL und den Host. Darauf folgen einige Headerfelder mit Informationen über den Client, die beispielsweise beschreiben, welche Datentypen der Client annimmt oder welchen Browser er verwendet.

3.3.1.1 Request-Methoden

Will ein Client eine Request an den Server schicken, gibt es eine Auswahl an Methoden. Einige davon sind seit der erstmaligen Entwicklung von HTTP veraltet und werden nicht mehr genutzt, da sie zu Sicherheitslücken führen.

Die am häufigsten genutzte HTTP-Methode ist die GET-Methode. Bei einem normalen Aufruf der GET-Methode fragt der Client beim Server nach einer bestimmten Ressource, die er erhalten möchte. Jedoch können mit der GET-Methode auch Parameter in Form von Querys in der URL mitgeschickt werden. Die dadurch übertragenen Daten sollten aber keine sicherheitsrelevanten Informationen enthalten, da sie direkt in der URL stehen. Darüber hinaus sollte eine GET-Methode nie Daten am Server verändern, sondern nur abrufen. Grund dafür, abgesehen davon, dass es so im Protokoll von HTTP definiert ist, ist, dass durch Prefetching eines Browsers oder ähnliche Prozesse, die GET aufrufen, gravierende, ungeplante Nebeneffekte auftreten können.

Definition 2.2: Prefetching

Im Bezug auf Web-Kommunikation ist Prefetching der Prozess, beliebig viele Hyperlinks einer geladenen Webseite auch zu laden, um dem Nutzer die Illusion einer schnelleren Internetverbindung zu geben, da wenn er einen Link nutzt, die Webseite dahinter bereits geladen wurde.

Sollen Daten am Server geändert werden, wird die POST-Methode oder die PUT-Methode genutzt.

Die POST-Methode schickt Daten an die URI, die dann vom Server verarbeitet werden. Der Unterschied zur PUT-Methode besteht darin, dass bei POST der Server die Daten erhält und dann entscheidet, wie er sie verarbeitet.

Wird PUT verwendet wird die übertragene Ressource, soweit möglich, direkt an die in der URI angegebene Stelle gespeichert. Ist dort bereits etwas gespeichert, wird es von der neuen Ressource überschrieben. Da dies ein zu starker Eingriff auf die Daten des Servers ist, haben heutzutage die meisten Server diese Funktionalität ausgeschaltet. Ebenso wird die Methode DELETE mittlerweile nicht mehr verwendet, da sie dazu dient Ressourcen auf dem Server zu löschen, was ähnlich viel Schaden anrichten kann wie die Fähigkeit mit PUT Ressourcen zu überschreiben.

3.3.2 Response-Nachricht

Hat der Server eine Request von einem Client erhalten und verarbeitet, schickt er eine Response-Nachricht zurück. Da die Verarbeitung einer Request nicht immer erfolgreich ist, sind verschiedene, standardisierte Statuscodes definiert, die der Server in der ersten Zeile seiner Response setzen kann. Diese sind aufgeteilt in fünf Kategorien:

- 1xx - Informationen
- 2xx - Erfolgreiche Anfrage
- 3xx - Umleitung
- 4xx - Client-Fehler
- 5xx - Server-Fehler

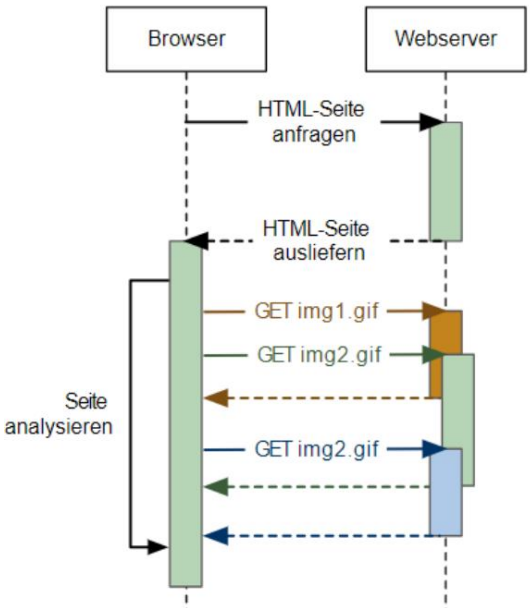
Die wichtigsten Statuscodes sind: 200 OK, alles wurde erfolgreich erledigt und die Antwort wurde gesendet; 301 Moved Permanently, die angefragte Ressource ist nicht mehr an dem Ort, wo sie der Client angegeben hat; 304 Not Modified, die angefragte Ressource befindet sich noch im Cache des Clients und wurde seit dem letzten Laden nicht verändert, wird also aus dem Cache geladen; 400 Bad Request, die Anfrage des Clients war fehlerhaft und konnte nicht verarbeitet werden; 401 Unauthorized, der Client ist nicht authentifiziert; 403 Forbidden, der Client ist nicht berechtigt diese Anfrage an diesen Server zu stellen; 404 Not Found, die angefragte Ressource wurde nicht gefunden; 500 Internal Server Error, im Server ist ein unerwarteter Fehler aufgetreten.

Je nach Status schickt der Server in seiner Response die angefragte Ressource oder, falls ein Fehler aufgetreten ist, Daten, die beim Lösen des Problems helfen. Beispielsweise schickt der Server bei Statuscode 301 Moved Permanently die neue Adresse im „Location“-Header-Feld mit sowie er bei 401 Unauthorized die Authentifizierungsmethode im „WWW-Authenticate“-Header-Feld mitschickt.

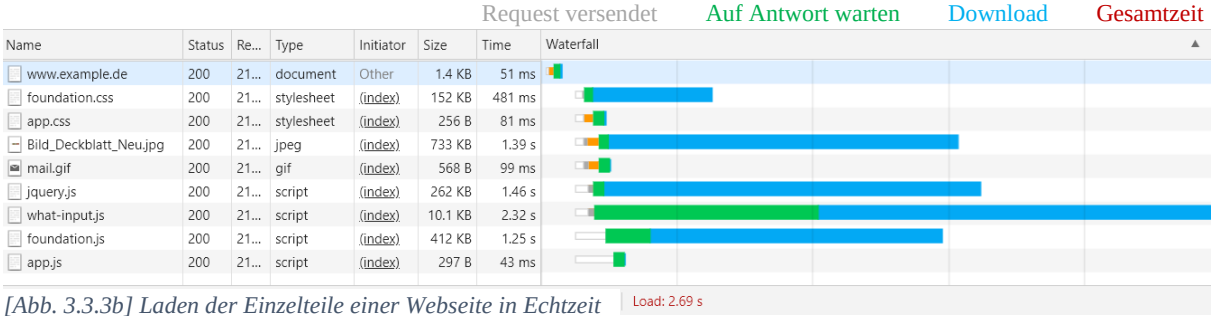
3.3.3 Laden einer HTML-Seite

Hat der Client ein HTML-Dokument durch eine GET-Anfrage beim Server erhalten, beginnt er diese zu interpretieren und analysieren. Daraufhin werden weitere Ressourcen per GET angefragt, auf die im HTML-Dokument verwiesen werden.

Dazu gehört fast immer eine oder mehrere CSS-Dateien und einige Bilder oder Skripte. Diese werden parallel geladen, sobald sie in der Analyse der HTML-Datei entdeckt werden. Werden bei der Analyse der CSS-Datei weitere Bilder gefunden werden auch diese per GET angefragt und geladen. Um den parallelen Download mehrerer Dateien zu gewährleisten, öffnet der Client zusätzliche Ports für jeden Download, an die der Server die angefragten Daten schickt. Der allgemeine Ablauf kann in Abb. 3.3.3a betrachtet werden. Abb. 3.3.3b zeigt wie der Prozess in Echtzeit abläuft.



[Abb. 3.3.3a] Sequenzdiagram des Ladens einer Webseite



[Abb. 3.3.3b] Laden der Einzelteile einer Webseite in Echtzeit

3.4 Sessions

Da HTTP und HTTPS zustandslose Protokolle sind, sich also keine Informationen über Clients merken, muss man den Zustand eines Clients auf andere Weise speichern. Die womöglich einfachste Lösung wäre per Querys in der URL. Diese Variante ist jedoch sehr unsicher, da sogar der User selbst seine Session-Daten verändern könnte.

Eine gute Lösung speichert also alle Daten und lässt den User nicht eingreifen. Um diese Voraussetzungen umzusetzen werden Session-Daten auf dem Server gespeichert. Verbindet sich eine Client zum ersten Mal mit der Webseite, wird ihm eine Session-ID zugewiesen. Diese wird zusammen mit jeglichen Daten zur Session auf dem Server gespeichert. Dem Client wird lediglich seine Session-ID mitgeteilt, mit der er sich bei der nächsten Interaktion mit dem Server identifiziert und somit vom Server mit seinen Daten in Verbindung gebracht werden kann.

Die meisten Sessions sind jedoch zeitlich begrenzt. Dies liegt einerseits daran, dass dann der Speicher für neue Sessions freigegeben werden kann. Andererseits gibt es sicherheitskritische Gründe, Sessions nicht endlos laufen zu lassen. Angreifer haben so weniger Zeit per Brute-Force oder ähnlichen Angriffsmethoden auf Serverdaten zuzugreifen. Dies schützt sowohl Server als auch andere Clients, deren Informationen gestohlen werden könnten. Es gibt zwei verschiedene Arten von Session Timeouts. Je nach Implementierung wird der Client nur ausgeloggt, wenn keine Aktivität mehr zu beobachten ist, oder es ist eine feste Zeit gesetzt, nach deren Ablauf die Session beendet wird. Ersteres ist für den User angenehmer, da, solange er die Webseite nutzt, er nicht auf die Zeit achten muss. Jedoch ist ein fester Session-Timeout sicherer und wird somit meist zusätzlich genutzt.

Zusammenfassung

In diesem Kapitel wurde zunächst das Protokoll zur Webkommunikation HTTP vorgestellt. Anhand der Nachteile von HTTP wird die Notwendigkeit der Sicherheitsverbesserungen in HTTPS erklärt.

Daraufhin wurde in weiterem Detail vorgestellt, wie die Kommunikation zwischen Server und Client abläuft. Es wurden wichtige Begriffe des Protokolls geklärt und Vor- und Nachteile der verschiedenen HTTP-Methoden abgewogen. Darauf folgt der Ablauf beim Laden einer HTML-Seite, um klar zu machen wie das Protokoll in der Praxis abläuft.

Zuletzt wurde auf die Zustandslosigkeit des HTTP-Protokolls eingegangen und erläutert, wie Server diese umgehen. Es wurde erklärt wie Sessions umgesetzt werden und welche Risiken dabei entstehen.

Aufgabe zur Selbstüberprüfung

Aufgabe 3.1

Wieso ist HTTP nicht sicher?

Aufgabe 3.2

Aus welchen zwei Hauptbestandteilen besteht eine HTTP-Nachricht und was ist in ihnen enthalten?

Aufgabe 3.3

Nennen Sie drei HTTP-Methoden und erläutern Sie was diese tun.

Aufgabe 3.4

Wie werden Sessions von den meisten Servern umgesetzt?

A Lösungen der Übungsaufgaben

Netzwerkadressierung

Aufgabe 1.1

IPv4: beispielsweise `206.34.0.168`

IPv6: beispielsweise `03c4:8a35:0003:0000:0000:003d:0000:0000`

Aufgabe 1.2

Port 80

Aufgabe 1.3

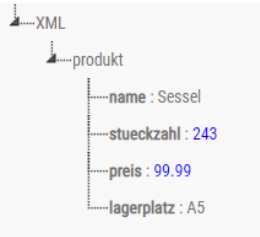
Schema, Authority, Path, Query, Fragment

Aufgabe 1.4

Query: ?username=hschneider&pin=1234

Datenstrukturen

Aufgabe 2.1



Aufgabe 2.2a

```
<!DOCTYPE produkt
[
  <!ELEMENT produkt (name, stueckzahl, preis, lagerplatz)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT stueckzahl (#PCDATA)>
  <!ELEMENT preis (#PCDATA)>
  <!ELEMENT lagerplatz (#PCDATA)>
]>
```

Aufgabe 2.2b

```
<xs:complexType name="produkt">
    <xs:sequence>
        <xs:element name="name"           type="xs:string"/>
        <xs:element name="stueckzahl"     type="xs:integer"/>
        <xs:element name="preis"          type="xs:decimal"/>
        <xs:element name="lagerplatz"     type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

Aufgabe 2.3

```
{
    "produkt": {
        "name":             "Sessel",
        "stueckzahl":       243,
        "preis":            99.99,
        "lagerplatz":       "A5"
    }
}
```

Client-Server-Kommunikation

Aufgabe 3.1

Wird HTTP als Protokoll genutzt, werden alle Daten unverschlüsselt als Klartext übertragen. Dadurch kann jeder, der im selben Netzwerk befindet, beliebig Daten mitlesen oder verändern.

Aufgabe 3.2

Head/Kopf: Im Kopf werden Metadaten übertragen.

Body/Körper/Rumpf: Im Rumpf werden die eigentlichen Daten übertragen.

Aufgabe 3.3

GET:	Fragt beim Server eine Ressource an.
POST:	Übergibt dem Server eine Ressource zum Verarbeiten.
PUT:	Speichert eine Ressource an die in der URI angegebene Stelle.
DELETE:	Löscht eine Ressource des Servers.

Aufgabe 3.4

Der Server gibt jedem Client eine Session-ID und speichert diese mit den Session-Daten des Clients ab. Jede Anfrage, die am Server ankommt, kann dann anhand der Session-ID mit einem Client in Verbindung gebracht werden.

B Literaturverzeichnis

[HIP4A] E. Ahlers, IPv4-Adress-Pool in Nordamerika endgültig erschöpft | heise online, <https://www.heise.de/newsticker/meldung/IPv4-Adress-Pool-in-Nordamerika-endgueltig-erschoepft-2826871.html> (19.11.2019)

[HIP4E] D. AJ Sokolov, Das war's mit IPv4-Adressein in Europa | heise online, <https://www.heise.de/newsticker/meldung/Das-war-s-mit-IPv4-Adressen-in-Europa-4596197.html> (1.12.2019)

[RFCTCP] M. Cotton, L. Eggert, J. Touch, M. Westerlund, S. Cheshire, RFC6335 - Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry, ISSN 2070-1721, < https://tools.ietf.org/html/rfc6335#section-6> (21.11.2019)

[RFCHF] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1, < https://tools.ietf.org/html/rfc2616#section-14> (30.11.2019)

[ITSM] H. Kersten, G. Klett, Der IT Security Manager, 4. Auflage, Springer Vieweg, Wiesbaden, 2015, ISBN 978-3-658-09973-2, ISBN (eBook) 978-3-658-09974-9

[RFCTLS] E. Rescorla, RFC 8446 – The Transport Layer Security (TLS) Protocol Version 1.3, < https://tools.ietf.org/html/rfc8446#section-1> (1.12.2019)

C Abbildungsverzeichnis

[Abb. 1.3.1a] T. Berners-Lee, R. Fielding, L. Masinter, RFC3986 – Uniform Resource Identifier (URI): Generic Syntax, <https://tools.ietf.org/html/rfc3986#section-3> (29.11.2019)

[Abb. 3.1.2] HTTP-Anfrage.svg <https://upload.wikimedia.org/wikipedia/commons/f/f9/HTTP-Anfrage.svg> (27.11.2019)

[Abb. 3.3.3a] F. Dopatka, Grundlagen der Web-Technologien (1.12.2019)