Basiswissen

Nachdem wir einen grundlegenden Überblick über die historische Entwicklung von Webapplikationen und deren Sicherheit gewonnen haben und uns ihrer Relevanz bewusst sind, nehmen wir uns nun der technischen Grundlagen an.

Wie in der Einleitung bereits erwähnt, ist dieses Kapitel primär dazu da, Klarheit über Begrifflichkeiten zu schaffen und sicherzustellen, dass von einem gleichen Verständnis ausgegangen wird. Es ist sinnvoll, sich die Grundlagen des Webs einmal vertiefend anzuschauen, um später die Beschreibung der Schwachstellen komplett nachvollziehen zu können.

Wir werden uns zunächst mit dem HTTP-Protokoll und seinen Eigenheiten beschäftigen, dann mit den wichtigsten Sprachen des Webs, mit URLs, mit Webbrowsern und abschließend mit möglichen Codierungen.

HTTP

HTTP (Hypertext Transfer Protocol) ist das grundlegende Protokoll des modernen Webs. Obwohl es ursprünglich nur für den Versand von statischen Texten gedacht war, basieren auf diesem Protokoll heute sämtliche modernen Webapplikationen.

Die erste Version von HTTP (HTTP/1.0) wurde 1996 im RFC 1945 [9] veröffentlicht und im Laufe der Zeit nur geringfügig verändert. Mit HTTP/1.1 aus dem Jahr 1999 (RFC 2616 [10]) schuf man einen Standard, der sogar über 15 Jahre Bestand hatte. Anfang 2015 wurde HTTP/2 als neueste Version des HTTP-Protokolls von der IETF (Internet Engineering Task Force) bekannt gegeben und in RFC 7540 [11] veröffentlicht.

Wir werden uns an dieser Stelle mit der modernsten Version des HTTP-Protokolls beschäftigen und die wichtigsten Aspekte in puncto Websicherheit betrachten.

Zunächst sei gesagt, dass HTTP zur Kommunikation zwischen Client (Anfragendem) und Server (Antwortendem) bzw. Rechnernetzen dient, deshalb kann man zwischen HTTP-Anfrage (Request) und HTTP-Antwort (Response) differenzieren. Ein Protokoll stellt immer einen Standard dar, der wie folgt aussieht: Ein Client öffnet mittels einer Anfrage eine Verbindung, der Server beantwortet die Anfrage mit einer Antwort und schließt die Verbindung.

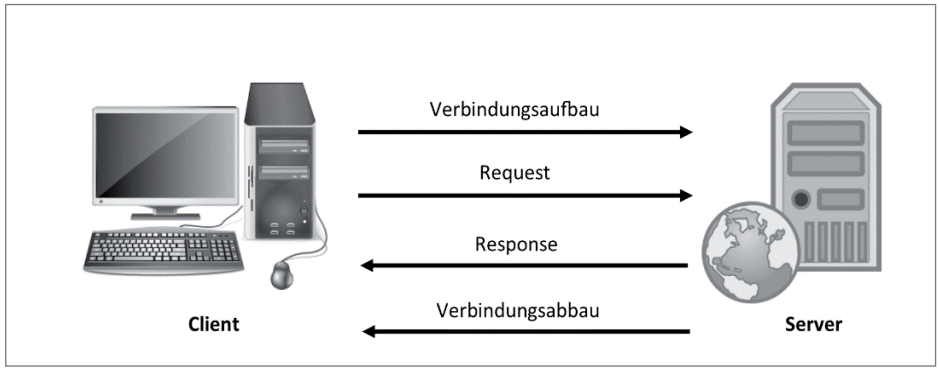

Bild 1: Ein typischer HTTP-Verbindungsablauf.

HTTP selbst baut auf einem Netzwerkprotokoll der Transportschicht auf, dem TCP (Transmission Control Protocol), über das die grundlegende Verbindung zustande kommt.

Grundsätzlich werden im HTTP-Protokoll alle Inhalte im Klartext übertragen, es besteht allerdings auch die Möglichkeit der Verschlüsselung mittels TLS (Transport Layer Security).

TLS UND SSL

Die Vorgängerversion von TLS ist SSL (Secure Sockets Layer). Oft werden die beiden Verschlüsselungsprotokolle synonym verwendet. TLS bietet jedoch entscheidende Vorteile und ist in jedem Fall das modernere und sicherere Verschlüsselungsverfahren.

Wenn eines der Verschlüsselungsprotokolle genutzt wird, um Inhalte mit HTTP zu übertragen, spricht man von HTTPS (Hypertext Transfer Protocol Secure). Dieses »sichere HTTP« sichert bei korrekter Verwendung die Vertraulichkeit und Integrität bei der Übertragung von Paketen durch Verschlüsselung. Auf die Möglichkeit der Verschlüsselung werden wir später erneut zu sprechen kommen. Wichtig ist an dieser Stelle nur, dass Nachfolgendes zwar standardmäßig unverschlüsselt übertragen wird (HTTP), dass aber auch eine verschlüsselte Übertragung (HTTPS) möglich ist.

Das eigentliche Protokoll und seine Befehle funktionieren bei beiden Arten der Übertragung gleich. Wir werden uns nun genauer damit beschäftigen und beginnen mit den zwei Arten von HTTP-Headern, die wir eben kennengelernt haben.

HTTP-Anfrage

Die sogenannte HTTP-Anfrage (HTTP-Request) wird vom Client zum Verbindungsaufbau genutzt und an den Server geschickt. Der Aufbau folgt einem Standard, damit sich sämtliche Clients und Server im Web verständigen können. Diesen Standard wollen wir im Folgenden einmal näher betrachten.

```
001 GET /test HTTP/2
002 Host: test.com
003 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0)
004 Accept: text/html,application/xhtml+xml,application/
    xml;q=0.9,*/*;q=0.8
005 Accept-Language: de,en
```

Die HTTP-Anfrage stellt eine Art HTTP-Kopf (HTTP-Header) dar.

HTTP-Header

Ein einzelner Parameter wird im täglichen Gebrauch häufig fälschlicherweise auch als HTTP-Header bezeichnet, korrekter ist jedoch die Bezeichnung HTTP-Headerfeld. Dadurch kann zwischen dem reinen Header, also der gesamten HTTP-Anfrage (siehe oben), und den einzelnen Werten und Parametern im HTTP-Header, den HTTP-Headerfeldern, differenziert werden.

Werfen wir nun einen genaueren Blick auf die HTTP-Anfrage und die entsprechenden HTTP-Headerfelder: Meist wird zunächst ein Parameter aufgeführt, und nach dem Doppelpunkt folgen die entsprechenden Zuweisungen eines Werts. Durch neue Zeilen werden die Werte voneinander getrennt, mehrere Werte in einer Zeile werden mit Kommas voneinander getrennt.

In unserem Beispiel sind folgende HTTP-Headerfelder beschrieben (vergleiche HTTP-Anfrage oben):

- Die erste Zeile ist die sogenannte Anfragezeile (Start-Line). Sie beginnt mit dem Aufruf einer Methode. (Was es mit den Methoden genau auf sich hat und welche es gibt, erfahren Sie später.)
- Es folgt eine Pfadbeschreibung. Sie enthält den Pfad, auf den die Methode angewendet werden soll. In unserem Fall wird die `GET`-Methode genutzt, um den Pfad `/test` aufzurufen.
- Dann folgt die verwendete HTTP-Version, in unserem Beispiel ist das der moderne Standard `HTTP/2`.

- Die zweite Zeile beschreibt den Host, der die abzurufende URL (siehe Kapitel 3.3, »URL«) umfasst. In der oben angeführten HTTP-Anfrage wird die URL `test.com` angefragt.
- Es folgt das HTTP-Headerfeld `User-Agent`, das die Art des Clients an die Applikation übermittelt. Zumeist wird darin der Browsertyp gesendet, etwa um Kompatibilitätsproblemen vorzubeugen. Das oben angeführte Beispiel ist also eine Anfrage von einem Mozilla Firefox-Webbrowser.
- Das `Accept`-Headerfeld beschreibt, welche Daten vom Client gelesen werden können, in unserem Fall beispielsweise Text- oder HTML-Dateien.
- Abschließend folgt das `Accept-Language`-Headerfeld, in dem angegeben wird, welche Sprache gelesen werden kann. Das ist wichtig, da sich jeder Sprache gewisse Zeichensätze zuordnen lassen.

Die oben angeführte Anfrage stellt ein simples Beispiel dar, eine HTTP-Anfrage kann sehr viele weitere Werte enthalten, die wir nach der Behandlung der HTTP-Antwort noch einmal in Augenschein nehmen werden.

HTTP-Antwort

Werfen wir nun einen genaueren Blick auf die HTTP-Antwort. Sie wird vom Server geschickt, nachdem eine bestimmte HTTP-Anfrage, etwa die oben angeführte, eingegangen ist.

```
001 HTTP/2 200 OK
002 Server: nginx
003 Date: Sun, 8 Mar 2015 12:36:19 GMT
004 Content-Type: text/html; charset=iso-8859-1
005 Vary: Accept-Encoding
006 Last-Modified: Tue, 09 Dec 2014 20:37:04 GMT
007 Connection: close
008 Content-Leight: 1237
009
010 <html>
011 <head>
012 <title>Testseite</title>
013 </head>
014 <body>
015 ...
```

Wie zu erkennen ist, gleicht der Aufbau einer HTTP-Antwort strukturell sehr der einer HTTP-Anfrage, allerdings gibt es einige neue Elemente:

- Die HTTP-Antwort beginnt in der ersten Zeile, der sogenannten Statuszeile (Response-Line), mit der vom Server verwendeten HTTP-Version.
- Danach folgt direkt ein dreistelliger Statuscode, in unserem Fall `200`, der dem Client signalisiert, ob es irgendwelche Probleme seitens des Servers gab oder ob Übertragung und Verarbeitung ohne Probleme funktioniert haben. (Die Bedeutung dieser Codes schauen wir uns gleich noch genauer an, siehe Kapitel 3.1.5, »Statuscode und Reason Phrase«.)
- Das `OK` am Ende der ersten Zeile ist die Beschreibung des zuvor mit dem Code beschriebenen Status und wird als »Reason Phrase« bezeichnet.
- In der zweiten Zeile, dem HTTP-Kopffeld `Server`, folgen Informationen über die Software bzw. das Betriebssystem des Servers, der die Anfrage verarbeitet hat.
- `Date` meldet das Datum des Servers an den Client.
- `Content-Type` beschreibt, welche Art von Datei übermittelt wurde.
- `Content-Leight` beschreibt, wie groß die Nachricht ist.

Nach allen HTTP-Headerfeldern folgt in der HTTP-Antwort eine Leerzeile und anschließend die eigentlich übermittelte Datei bzw. Nachricht (Message-Body). In unserem Fall ist es der Anfang einer Webseite, die dann vom Client interpretiert wird (siehe dazu Kapitel 3.4, »Webbrowser**«**).

Als normaler Nutzer sieht man die Headerfelder normalerweise nicht, dennoch sind sie essenziell für das HTTP-Protokoll und die Benutzung des Webs, denn über sie werden die Schlüsselinformationen zwischen Client und Server kommuniziert.

HTTP-Headerfelder

Wir haben soeben gelernt, dass sowohl HTTP-Anfragen als auch HTTP-Antworten im HTTP-Header bestimmte HTTP-Headerfelder mit Werten aufweisen.

Es gibt grundlegende Headerfelder und spezifische, manche gelten für Anfragen und Antworten, manche allerdings auch nur für eines von beiden. Es gibt also eine Vielzahl von Headern. Im Laufe der Zeit wurden allerdings auch viele in ihrer Entwicklung eingestellt, zurückgezogen oder sind wahre Exoten.

Nachfolgend werden wir eine Auswahl der wichtigsten Headerfelder betrachten, die oben noch nicht aufgeführt wurden, aber Relevanz haben. Alle »nicht standardisierten« Header haben ein `X-` am Anfang. Die meisten Header werden uns sicherlich auch noch an der einen oder anderen Stelle im Buch begegnen. Die Unterstützung der Headerfelder ist vom benutzten Webbrowser abhängig.

Anfrage-Headerfelder

`Accept-Encoding`: Beschreibt, welche Art von Codierung für den Client lesbar und zulässig ist.

`Cookie`: Übermittelt einen Textwert an den Server, der vorher in einer HTTP-Antwort mittels `Set-Cookie` gesetzt werden kann. Wozu das dient, werden wir im Kapitel über Cookies (siehe Kapitel 3.1.6, »Cookies«) untersuchen.

`Host`: Beschreibt, welche Ressource angefragt wird, meist handelt es sich um eine URL (siehe Kapitel 3.3., »URL«).

`Referer`[1]: Enthält die URL, auf der der Nutzer vor dem Besuch der nun angefragten Seite war.

Antwort-Headerfelder

`Expires`: Teilt dem Client mit, wie lange der übermittelte Inhalt seine Gültigkeit behält.

`Set-Cookie`: Bildet das Pendant zum Cookie in einer HTTP-Anfrage und bietet die Möglichkeit, einen Wert für dieses HTTP-Headerfeld zu setzen.

`X-Frame-Options`: Macht Angaben darüber, ob die Webseite eingebettet werden darf (siehe Kapitel 11.2.2, »**Fehler! Verweisquelle konnte nicht gefunden werden.«**).

`X-XSS-Protection`: Bietet die Möglichkeit, einen Schutz vor XSS im Browser zu aktivieren (siehe Kapitel 5.9.2, »**Fehler! Verweisquelle konnte nicht gefunden werden.«**).

`Content-Security-Policy`: Kurz CSP genannt, dabei handelt es sich um die Möglichkeit, festzulegen, welche Inhalte innerhalb der Webseite noch nachgeladen werden dürfen. Es ist also eine zusätzliche Sicherheitsmaßnahme, auf die wir später genauer zu sprechen kommen (siehe die Kapitel 5.9.1, »**Fehler! Verweisquelle konnte nicht gefunden werden.«** und 11.2.1, »**Fehler! Verweisquelle konnte nicht gefunden werden.«**).

`Content-Encoding`: Inhalte im Web werden häufig komprimiert übertragen, um Bandbreite zu sparen. Oft kommt dabei das »gzip-Verfahren« zum Einsatz, das Inhalte ohne Verlust komprimieren kann.

---

[1] Es handelt sich dabei tatsächlich um den Begriff »Referer«, korrekterweise müsste er jedoch »Referrer« geschrieben werden. Dieser Rechtschreibfehler hat sich in eine der ursprünglichen HTTP-Beschreibungen eingeschlichen und wurde bis heute nicht behoben.

Generelle Headerfelder

`Connection`: Dient primär dazu, den Status oder die Art der Verbindung festzulegen, am Ende einer Antwort steht häufig `Connection: close`, um die entsprechende Verbindung zu schließen.

HTTP-Methoden

Wie im Abschnitt über HTTP-Anfragen bereits erwähnt, werden sogenannte HTTP-Methoden genutzt, damit der Client dem Server mitteilen kann, wie die Daten angefordert werden. Die Methoden und ihre Eigenarten sind im RFC 2616 [10] näher erklärt. Jede Anfrage beginnt in der Start-Line mit einer Methode.

Die Methode `GET` kann ein Dokument anfordern. Dieses kann sowohl statisch auf dem Server liegen als auch dynamisch, zum Beispiel aus verschiedenen Quellen, generiert werden. Die `GET`-Methode wird in den Logdateien eines Webservers protokolliert. `GET` ermöglicht eine Übertragung von Parametern direkt in der URL. Das eben kennengelernte HTTP-Headerfeld `Referer` wird bei der Methode `GET` an die neu aufgerufene Ressource weitergegeben. Aus diesem Grund sollten niemals sensible Informationen direkt in der URL enthalten sein.

Eine andere oft genutzte Methode ist `POST`, die zum Beispiel für Formulare verwendet wird und immer dann, wenn der Nutzer eine Aktion ausführt. `POST` wird im Gegenteil zu `GET` in den meisten Logdateien einer Webapplikation nicht erfasst und überträgt auch nicht den Referrer. Die `POST`-Methode ermöglicht die Übertragung der Parameter sowohl in der URL als auch in der Nachricht selbst.

`GET` und `POST` sind mit Abstand die am weitesten verbreiten HTTP-Methoden im Web, allerdings gibt es noch zahlreiche weitere. Die wichtigsten wollen wir nachfolgend betrachten.

`HEAD` ist im Grunde genommen exakt die gleiche Methode wie `GET`, der einzige Unterschied besteht darin, dass sich in der HTTP-Antwort kein Nachrichtenteil befindet, sondern ausschließlich der HTTP-Header übertragen wird.

`OPTIONS` ermöglicht es, herauszufinden, welche HTTP-Methoden vom Server für die angefragten Ressourcen unterstützt werden. Meist wird das deutlich gemacht, indem serverseitig ein `Allow`-Headerfeld mitgesendet wird.

`TRACE` erlaubt dem Client, so anzufragen, dass dieser in der Lage ist, nachzuvollziehen, was der Server empfängt. Das funktioniert, indem die Anfrage als Antwort zurückgegeben wird. Der Server sendet also die empfangene Anfrage vollständig zurück und deklariert sie mittels des folgenden HTTP-Headerfelds: `Content-Type:"message/`

`http"`. Diese Methode eignet sich sehr gut für Diagnosezwecke, beispielsweise um herauszufinden, inwieweit ein Proxy die Anfrage verändert.

Statuscode und Reason Phrase

Wie wir in der HTTP-Antwort gesehen haben, enthält sie in der Statuszeile einen dreistelligen Statuscode. Diese Statuscodes sind ebenfalls im RFC 2616 [10] festgelegt und gliedern sich nach folgendem Schema:

1XX — Information

2XX — erfolgreicher Abruf

3XX — Weiterleitung

4XX — Error (beliebige Art)

5XX — Error (der die Lauffähigkeit der Applikation beeinflusst)

Insgesamt gibt es rund 42 Statuscodes mit Reason Phrases — die (aus Sicherheitssicht) wichtigsten neun sind nachfolgend angeführt.

Statuscode	Bedeutung (Reason Phrase)	Bedeutung
200	OK	Die Anfrage war erfolgreich, und es wurde eine Antwort mit Nachrichteninhalt übermittelt.
301	Moved Permanently	Leitet den Webbrowser dauerhaft auf eine andere Ressource (Location-Header) weiter.
302	Found	Leitet den Webbrowser temporär auf eine andere Ressource (Location-Header) weiter.
400	Bad Request	Die Anfrage des Clients wird vom Server als fehlerhaft verworfen.
403	Forbidden	Der Zugriff auf die angefragte Ressource ist untersagt.
413	Request Entity Too Long	Die Anfrage erzeugt eine zu lange Antwort, der Server muss sie aufgrund der Länge verwerfen.
414	Request URI Too Long	Ähnlich wie 413. Die genutzte URL kann aufgrund der Überlänge nicht vom Server verarbeitet werden.

Statuscode	Bedeutung (Reason Phrase)	Bedeutung
500	Internal Server Error	Der Request konnte nicht verarbeitet werden, die Applikation bricht die weitere Verarbeitung ab.
503	Service Unavailable	Die Applikation ist im Moment nicht verfügbar (der Server arbeitet normal).

Cookies

Wir haben bereits die HTTP-Headerfelder kennengelernt, dabei sind wir unter anderem auf die Headerfelder `Set-Cookie` und `Cookie` eingegangen. Bislang haben wir beides lediglich als Möglichkeit betrachtet, Textwerte zu setzen und zu übertragen.

Es wird serverseitig ein Textwert gesetzt (`Set-cookie`), der Client speichert ihn und sendet ihn bei erneuter Anfrage wieder zurück an den Server (`Cookie`).

Um zu verstehen, welcher Vorteil daraus entsteht, muss man ein wichtiges Prinzip des HTTP-Protokolls kennen: HTTP ist ein zustandsloses Protokoll.

Letzteres bedeutet, dass standardmäßig die einzelnen Anfragen eines Clients nicht im Zusammenhang gesehen werden. Wenn man also bei einem Server eine Unterseite anfordert und anschließend auf eine andere Unterseite wechselt, ist das höchstens an der IP-Adresse oder dem Referrer zu erkennen, sonst nicht. Eine wirkliche Differenzierung oder sogar Anpassung der Seite, da die vorherigen Schritte bekannt waren, kann aufgrund dieser Daten nicht erfolgen. Es ist nicht möglich, sie sicher einem konkreten Nutzer zuzuordnen, da es heutzutage komplexe Systemstrukturen gibt, bei denen mehrere Clients eine IP-Adresse haben können.

Ein Proxyserver (eine Kommunikationsschnittstelle mit einer oder einigen IP-Adressen) kann beispielsweise Tausende von Clients versorgen, wenn diese eine Webseite aufrufen. So würde es für den Server wie ein einzelner anfragender Client aussehen, und es gäbe massive Authentifizierungsprobleme. Dieser Problematik musste man entgehen, da eine dauerhafte Authentifizierung auf Applikationsebene sonst nicht umsetzbar gewesen wäre. Um Clients nach einer Anfrage wieder erkennbar zu machen und Sitzungen dauerhaft aufrechterhalten zu können, entwickelte man also das Konzept der Cookies.

Durch diese Spezifikation, die erst 1994 Einzug in das Web hielt, war es gelungen, die Zustandslosigkeit des HTTP-Protokolls zu überwinden.

Cookies finden bis heute vielfältigen Einsatz, insbesondere um Sitzungen zu ermöglichen. Welchen Zweck diese Sitzungen (Sessions) genau haben, ergründen wir im Kapitel über Session-Angriffe (siehe Kapitel 4, »Fehler! Verweisquelle konnte nicht gefunden werden.«).

Zunächst werfen wir jedoch noch einmal einen genauen Blick auf Cookies. Das Anlegen eines Cookies vom Server (über HTTP-Headerfelder) sieht wie folgt aus:

```
Set-Cookie: auth=172d046067681c6a5f5b7039434b868
```

Durch Setzen dieses Werts wird bei jeder weiteren Anfrage des Clients automatisch im Browser folgendes HTTP-Headerfeld hinzugefügt:

```
Cookie: auth=172d046067681c6a5f5b7039434b868
```

Statt eines Cookies können mit `Set-Cookie` auch mehrere gesetzt werden, allerdings werden sie dann in einem einzigen »Cookie-Headerfeld« vom Client wieder zurückgegeben. Getrennt werden die einzelnen Cookies mittels Semikolon (;).

Ein Cookie besteht immer mindestens aus einem Namen und einem Wert. Es gibt allerdings viele weitere Informationen, die beim Setzen des Cookies (`Set-Cookie`) hinzugefügt werden können, damit der Webbrowser genauer weiß, wie er mit einem Cookie umgehen soll. Es folgt eine Liste der wichtigsten Attribute für das HTTP-Headerfeld `Set-Cookie`:

- `expires` — Gibt genau an, wie lang ein Cookie gültig ist. Die Angabe hat folgende Gliederung: »Wochentag, Tag Monat Jahr `HH:MM:SS GMT`«.
- `discard` — Legt die unbedingte Löschung des Cookies nach Schließen des Browsers fest.
- `max-age` — Arbeitet ähnlich wie `expires`, allerdings wird die Laufzeit des Cookies hier in Sekunden übergeben.
- `domain` — Gibt an, für welche Domain das Cookie gilt. Jede Domain kann nur für sich selbst oder entsprechende Subdomains das Cookie setzen. Bleibt der Wert leer, wird automatisch der aktuelle Domainname verwendet.
- `path` — Ermöglicht es, näher anzugeben, für welchen Pfad das Cookie gelten soll.
- `port` — Gibt an, ob sich das Cookie auf einen bestimmen Port bezieht.
- `comment` — Ermöglicht die direkte Kommentierung des Cookies.
- `commenturl` — Ermöglicht das Hinterlegen einer URL, die die Funktionsweise des Cookies beschreibt.

- `secure` — Wenn dieser Wert beim Setzen eines Cookies aktiviert ist, wird das Cookie nur bei einer verschlüsselten Verbindung (HTTPS) gesendet.
- `HttpOnly` — Ist dieser Wert gesetzt, ist im Browser nur noch mittels HTTP ein Zugriff auf das Cookie möglich (über JavaScript und ähnliche Skriptsprachen ist kein Zugriff möglich).

Die Laufzeiten von Cookies sind insbesondere im Bereich Session-Management von Bedeutung (siehe Kapitel 4.6, »Fehler! Verweisquelle konnte nicht gefunden werden.«).

Viele der Regeln zum Setzen von Cookies spiegeln sich zudem in der SOP (Same-Origin-Policy) wider.

Cookies zurückrufen

Auch wenn in den Informationen des Cookies klar festgelegt wird, wann ein Cookie zu löschen bzw. als ungültig zu betrachten ist, heißt das nicht, dass der Client es auch tatsächlich löscht. Cookies sind Textdateien, ein Client kann sie beliebig ändern oder ihre Löschung verhindern. Das bedeutet, dass man als Administrator serverseitig das Cookie widerrufen muss und nicht darauf vertrauen darf, dass Cookies automatisch gelöscht werden. Die Manipulation von Cookies durch Angreifer kann gravierende Folgen haben, wenn der Entwickler einfache Grundsätze nicht berücksichtigt.

Abschließend betrachten wir zum Verständnis noch einmal den Ablauf einer HTTP-Anfrage und einer HTTP-Antwort mit Cookie-Setzung:

Erste Anfrage des Clients:

```
001 GET /index.html HTTP/2.0
002 Host: test.url
003 …
```

Serverantwort:

```
001 HTTP/2.0 200 OK
002 Content-Type: text/html
003 Set-Cookie: tracing=ce9197b30
004 Set-Cookie: auth=172d046067681c6a5f5b7039434b868; Path=/accounts;
    Expires=Wed, 15 Jan 2020 12:37:49 GMT; Comment=Testcookie; Secure;
    HttpOnly
005 …
```

Erneute Clientanfrage (mit gesetztem Cookie):

```
001 GET /index.html HTTP/2.0
002 Host: test.url
003 Cookie: tracing=ce9197b30; auth=172d046067681c6a5f5b7039434b868
004 …
```

Die wichtigsten Funktionen von Cookies sollten nun deutlich geworden sein. Sie werden uns aufgrund ihrer Wichtigkeit an einigen weiteren Stellen in diesem Buch begegnen, dann gehen wir auch genauer auf den Sinn der oben angeführten Flags (`Secure` und `HttpOnly`) ein.

Sprachen des Webs

Die programmierte Logik einer Webanwendung ist aufgeteilt zwischen Client und Server. Für die Ausführung von Programmen auf dem Client-Rechner werden meistens andere Sprachen benutzt als auf dem Server. Das liegt auch daran, dass die Sprache auf dem Client in der abgeschotteten Umgebung des Browsers abläuft und nicht direkt auf Betriebssystemebene wie bei einem Server. Deswegen muss man sich als Webentwickler mit unterschiedlichen Sprachen auseinandersetzen und diese auch gegen Angriffe absichern.

Clientseitige Sprachen

Bei clientseitigen Sprachen handelt es sich um Sprachen, die anwenderseitig interpretiert bzw. ausgeführt werden. Eine klassische Umgebung, in der diese Sprachen interpretiert werden, ist ein Webbrowser. Mit Browsern beschäftigen wir uns in Kapitel 3.4, »Webbrowser« näher.

HTML

Das »Grundgerüst« von Webseiten ist in HTML (Hypertext Markup Language) geschrieben. Dabei handelt es sich um eine abstrakte Sprache, die die Struktur der übermittelten Website beschreibt. Der erste Entwurf von HTML beruhte noch weitgehend auf der Standard Generalized Markup Language (SGML), einer Metasprache von 1986 (ISO 8879 [12]), die damit bereits deutlich älter als das Web ist. Der Entwurf zu HTML stammt von Tim Berners-Lee aus dem Jahr 1989 [2], in der Zwischenzeit wurden viele weitere Versionen von HTML veröffentlicht. Der moderne Standard ist HTML5, der jedoch bisher von den verschiedenen Webbrowsern unterschiedlich gut unterstützt wird.

Der Aufruf solcher HTML-Dokumente funktioniert nach folgendem Prinzip: Ein Benutzer lädt mittels HTTP-Anfrage ein HTML-Dokument herunter (siehe HTTP-Anfrage oben), und der Inhalt wird nach Interpretation durch einen Webbrowser auf dem Endgerät des Benutzer angezeigt.

Bis heute hat sich HTML als Standard gehalten, sämtliche Webseiten nutzen HTML, um eine Auszeichnung und die Beschreibung der einzelnen Elemente vorzunehmen. Konkret werden mit HTML die Struktur und die Semantik weitestgehend festgelegt. Prinzipiell wäre die Darstellung der Elemente auch ohne HTML denkbar; so lässt sich auch einfacher Text ausgeben (`txt`), allerdings ohne dass Bilder und sonstige Medien eingebunden werden können.

HTML bietet die Möglichkeit, Elemente verschiedener Art strukturiert auszugeben, und dient somit als Ausdruckssprache.

HTML ist stets nach demselben Schema gegliedert, das wie folgt aussieht:

```
001 <html>
002   <head>
003     <title>Titel der Webseite</title>
004   </head>
005   <body>
006    <h1>Überschrift</h1>
007    <p>Inhalt der Webseite</p>
008    <b>fett gedruckter Text</b>
009   </body>
010 </html>
```

Diese HTML-Informationen werden einfach an den HTTP-Header der Serverantwort angehängt (siehe Kapitel 3.1.2, »HTTP-Antwort«) und dadurch an den Benutzer übertragen.

Ein Element, zum Beispiel ein Text, ist von sogenannten HTML-Tags eingeschlossen. HTML-Tags stehen in spitzen Klammern (< >) und beschreiben, wie der Inhalt interpretiert werden soll. Es gibt jeweils ein öffnendes Tag (<tag>) und ein schließendes Tag (</tag>), Letzteres ist immer mit einem Slash (/) als zweites Zeichen gekennzeichnet.

Ein HTML-Dokument beginnt stets mit einem öffnenden HTML-Wurzelelement, `<html>`, und endet mit einem schließenden Element, `</html>`.

Zwischen diesen HTML-Wurzeltags befinden sich der HTML-Header (gekennzeichnet mit `<head></head>`) und der sogenannte Body (gekennzeichnet mit `<body></body>`).

Wie die Namen andeuten, handelt es sich um besondere Bereiche, die verschiedene Elemente enthalten können:

- Der HTML-Header enthält grundlegende Informationen zum nachfolgenden Inhalt. In unserem Codebeispiel wird darüber der Seitentitel gesetzt (siehe oben). Es gibt allerdings eine Vielzahl weiterer Elemente, die in einem HTML-Header enthalten sein können, etwa eingebundene externe Dateien oder sogenannte Metatags, in denen generelle Spezifikationen an der HTML-Datei vorgenommen werden können (etwa der verwendete Zeichensatz oder weitere grundlegende Informationen zur Webseite).
- Der HTML-Body kann sehr viele verschiedene Arten von Elementen enthalten, etwa Texte, Bilder, Videos und vieles mehr — kurz gesagt: alles, was uns später als Inhalt ausgegeben wird.

Folgendes Beispiel zeigt anhand eines `<b>`-Tags, wie ein HTML-Code interpretiert wird:

```
<b>fett gedruckter Text</b>
```

fett gedruckter Text

Wie man in diesem Beispiel sehen kann, erzeugt das Einschließen durch sogenannte Bold-Tags Fettdruck.

Links

Des Weiteren können über HTML (Hyper-)Links gesetzt werden. Sie sind folgendermaßen aufgebaut, und man unterscheidet zwischen zwei Typen:

```
001 <!-- Kommentar: absoluter Link -->
002 <a href="http://test.url/testverzeichnis/index.html">
    Testverzeichnis</a>
003 <!-- Kommentar: relativer Link -->
004 <a href="/testverzeichnis/index.html">Testverzeichnis</a>
```

Beide erzeugen einen Verweis auf die angeführte Ressource. Bei einem absoluten Link muss jedoch die gesamte URL und nicht nur ein Pfad eingefügt werden. Die Links werden beide als »Testverzeichnis« dargestellt (Links sind per Standard unterstrichen und in den meisten Webbrowsern blau, sofern die Webseite keine anderen Vorgaben macht und der Link noch nicht geklickt wurde).

Ein Klick auf jeden dieser Links führt zu folgender HTTP-Anfrage:

```
GET /testverzeichnis/index.html HTTP/2.0
Host: test.url
…
```

Bei relativen Links steht der Host in Abhängigkeit zur bereits aufgerufenen Webseite. In dem Beispiel oben wurde angenommen, dass wir bereits auf einer Seite von `test.url` waren.

Es gibt übrigens auch die Möglichkeit, Parameter an einen Link anzuhängen, aber damit beschäftigen wir uns erst im Unterkapitel über URLs näher (siehe Kapitel 3.3, »URL«).

Kommentare
In dem Codeauszug oben haben wir ein zusätzliches Element kennengelernt: Kommentare. Diese werden mit `<!--` eingeleitet und mit `-->` beendet. Kommentare werden für Menschen geschrieben und enthalten zusätzliche Informationen, die jedoch grundsätzlich keine Rolle bei der Gestaltung oder Strukturierung einer Webseite spielen.

Formulare
Ein weiteres sehr verbreitetes HTML-Element sind Formulare. In HTML sehen sie folgendermaßen aus:

```
<form action="http://test.url/cgi-bin/kommentar.pl" method=GET>
   <table border="0" cellpadding="5" cellspacing="0">
   <tr>
      <td align="right">Vorname:</td>
      <td><input name="Vorname" type="text" size="40" maxlength="40"
</td>
   </tr>
   <tr>
      <td align="right" valign="top">Kommentar:</td>
      <td><textarea name="Text" rows="10" cols="40"></textarea></td>
   </tr>
   <tr>
   <input type="submit" value="Absenden">
   <input type="reset" value="Abbrechen">
   </tr>
   </table>
</form>
```

Das hier dargestellte HTML erzeugt zwei Eingabefelder und zwei Buttons und wird in vergleichbarer Form in Kommentarformularen auf vielen Webseiten eingesetzt.

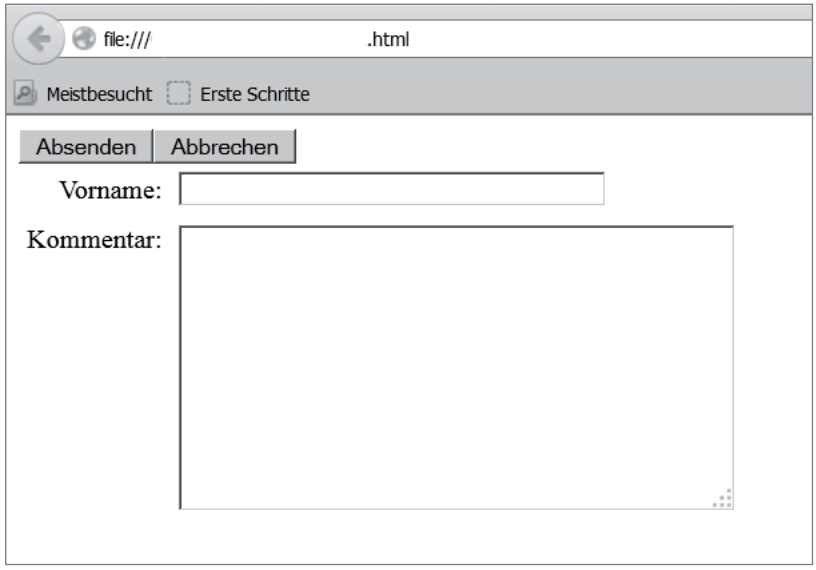

Bild 2: Darstellung des oben angeführten Codes**.**

In der ersten und der letzten Zeile befinden sich `form`-Elemente. Wie man der ersten Zeile des Codes entnehmen kann, wird mittels `action`-Attribut festgelegt, womit die Eingaben verarbeitet werden sollen und auch mit welcher Methode, in diesem Fall mittels `GET`. Die Daten werden an ein Perl-Skript übergeben, denkbar wäre aber auch die Verarbeitung mit einer Vielzahl weiterer Skriptsprachen. Die wichtigsten werden wir uns im Anschluss an den HTML-Abschnitt ansehen.

Manche der `input`-Tags besitzen kein End-Tag, sie werden als Stand-alone-Tags bezeichnet. Die Namen der Tags dienen der Ansprache durch Skriptsprachen und der Bezeichnung innerhalb der Webseite. Der Typ eines `input`-Tags (`type`) dient dazu, festzulegen, welche Daten eingegeben werden können, in diesem Fall beispielsweise `text`, um eine Texteingabe zu ermöglichen.

Es gibt allerdings noch viele weitere Möglichkeiten, die wichtigsten sind in der folgenden Tabelle zusammengefasst:

Wert	Beschreibung	HTML5
button	anklickbarer Button (meist in Kombination mit JavaScript)	
email	Feld zur Eingabe einer E-Mail	X
date	Datum	X
number	Feld zur Eingabe einer Nummer	X
password	Feld zur Eingabe eines Passworts (gleicht text, allerdings werden die Zeichen nicht dargestellt, sondern "ausgeschwärzt")	
radio	Optionsschaltfläche	
url	Feld zur Eingabe einer URL	X

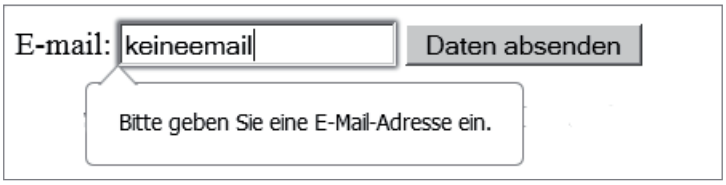

Bild 3: Clientseitige Validierung einer E-Mail-Adresse durch HTML5.

Seit HTML5 kann im Browser direkt überprüft werden, ob die eingegebenen Daten mit dem erwarteten Datentyp übereinstimmen. An dieser Stelle sei gesagt, dass solche clientseitigen Filter nicht die Sicherheit der Applikation stärken, sondern bloß verhindern, dass Nutzer versehentlich Anfragen senden, die nicht verarbeitet werden können. Ein Angreifer könnte dem Parameter immer noch nach Belieben Werte zuweisen — deshalb ist immer eine serverseitige Validierung vonnöten (dazu später mehr).

Frames

Frames bieten die Möglichkeit, Webseiten auf anderen Webseiten einzubinden. Sogenannte Framesets werden benutzt, um die Einbindung meist interner Seiten in einem definierten Bereich des Bildschirms vorzunehmen.

Unter dem Aspekt der Sicherheit erweisen sich eingebettete Frames (sogenannte Iframes) allerdings als deutlich brisanter. Denn durch Letztere ist es möglich, Webseiten positionsgenau einzubetten und sie innerhalb einer anderen Webseite zu laden.

Folgender Code auf einer Webseite bindet beispielsweise die Webseite des Franzis Verlags (*www.franzis.de*) in die betreffende Webseite mit ein:

```
001 <iframe src="http://www.franzis.de/">
002    <p>Dieser Browser unterstützt keine Iframes</p>
003 </iframe>
```

Bild 4: Darstellung eines Iframes in Mozilla Firefox.

Wir werden später (siehe Kapitel 11, »Fehler! Verweisquelle konnte nicht gefunden werden.«) sehen, dass die gezielte Einbindung von Webseiten durchaus zu Problemen führen kann.

Für alle uns im Web bekannten Elemente gibt es einzelne Tags und Attribute. Da dies allerdings kein HTML-Buch ist, soll uns diese Einführung erst einmal genügen.[2]

HTML bietet bis heute »die Basis des Webs«, über die Elemente aller Art strukturiert und zum Beispiel auch Formulare erzeugt werden. Die eigentliche Funktionalität und die Verarbeitung der eingegebenen Daten werden allerdings von den betreffenden Programmiersprachen übernommen. Vom Sicherheitsaspekt her ist das Verständnis von HTML unabdingbar. In einigen Kapiteln werden wir uns detaillierter mit der Möglichkeit einzelner HTML-Tags und ihrer Verschachtelung beschäftigen.

XML

Die Extensible Markup Language (XML) ist ebenfalls eine Auszeichnungssprache, sie basiert wie HTML auf der Standard Generalized Markup Language (SGML [12]).[3]

XML findet seit seiner Einführung im Jahr 1998 bis heute breite Anwendung. Diese Sprache bietet die Möglichkeit, Informationen strukturiert auszutauschen, und enthält wie HTML auch öffnende und schließende Tags, in denen sich Informationen befinden können.

---

2 Siehe Literaturempfehlung: »HTML5 Handbuch«.

3 SGML selbst wurde 2012 als Norm übrigens ersatzlos zurückgezogen.

```
<buch>
   <inhaltsverzeichnis>
      <kapitel1>Einleitung in das Kapitel</kapitel1>
      <unterkapitel></unterkapitel>
   </inhaltsverzeichnis>
</buch>
```

Im obigen Codebeispiel wird ein Auszug aus einem Buch dargestellt. XML wird im Web häufig verwendet, da sich darüber verschiedene Dateiinhalte simpel darstellen oder verarbeiten lassen. Dass genau bei dieser Verarbeitung Schwierigkeiten oder Probleme auftauchen können, untersuchen wir später genauer.

XHTML

Neben dem reinen HTML gibt es das sogenannte XHTML (Extensible Hypertext Markup Language), dabei handelt es sich, wie der Name schon gesagt, um eine Art Erweiterung von HTML. HTML wurde um die zuvor beschriebene Auszeichnungssprache XML ergänzt.

Wirklich erweitert wurde HTML dabei allerdings nicht, vielmehr wurde es standardisiert, um die einwandfreie Interpretation von Code zu ermöglichen. HTML selbst erlaubt beispielsweise sowohl die Groß- als auch die Kleinschreibung von Tags (`<h1>` und `<H1>` sind äquivalent).

Des Weiteren würde folgender Code in HTML weitgehend verstanden werden:

```
<b><i>Dieser Text ist fett und kursiv</b></i>
<p>Ich bin ein Absatz
<p>Ich bin der nächste Absatz
```

Laut XHTML ist dieses Codebeispiel jedoch falsch. Wir haben gelernt, dass Tags nach dem Öffnen immer erst geschlossen werden müssen, bevor neue Tags verwendet werden können (siehe Zeile 1). Zeile 2 und 3 wären ebenfalls syntaktisch falsch, da kein schließendes Tag verwendet wurde.

In XHTML würde dieser Code also verworfen werden, da zum einen eine Verschachtelung von Tags als unzulässig gilt und zum anderen jedes Tag klar geschlossen werden muss.

Ein korrekt umgesetzter XHTML-Code sähe also folgendermaßen aus:

```
<b><i>Dieser Text ist fett und kursiv</i></b>
<p>Ich bin ein Absatz</p>
<p>Ich bin der nächste Absatz</p>
```

XHTML wurde als Versuch entwickelt, einen Standard zu schaffen, den jeder Browser versteht. Mehr dazu erfahren wir, wenn wir uns später mit Browsern und HTML-Parsern beschäftigen.

CSS

CSS (Cascading Style Sheets, in Deutsch etwa »gestufte Gestaltungsbogen«) ist eine weitere wichtige Möglichkeit der Gestaltung von Webseiten. Während es bei HTML primär um die Struktur der Webseite geht, nutzt man CSS, um die platzierten Elemente zu gestalten. Der moderne CSS-Standard ist CSS 3, allerdings gibt es immer wieder weitere Entwicklungsschritte.[4]

CSS-Code sieht etwa folgendermaßen aus:

```
001 p.info span {
002    font-weight: bold;
003 }
```

Ein entsprechender HTML-Code, in dem er verwendet wird, könnte so aussehen:

```
001 <p class "info">
002    <span>Fettdruck<span>
003 </p>
```

Die oben angeführten Codeschnipsel würden zusammengefügt folgende Ausgabe erzeugen:

Fettdruck

CSS kann auf HTML-Elemente angewendet werden und diese gestalten. Häufig wird CSS verwendet, um eine Spezifizierung der Schrift, der Farben oder der Abstände zwischen Elementen in HTML vorzunehmen, die Ansprache der Elemente wird häufig über das bereits bei Input-Elementen kennengelernte `name`-Element vorgenommen. Man kann für eine Art von Elementen aber auch generell einen CSS-Code erstellen.

In dem oben angeführten Beispiel erhält das HTML-Element `info` eine Schriftgröße `bold`, also den Fettdruck, den wir bereits im HTML-Kapitel oben kennengelernt haben.

Es gibt zwei Möglichkeiten, CSS einzubinden:

---

[4] Offiziell wurde 2011 als Standard zunächst »CSS Level 2 Revision 1 (Recommendation)« eingeführt, aber CSS 3 wird bald ebenfalls zum Standard.
Literaturempfehlung: »Webseiten-Layout mit CSS: Der perfekte Einstieg in Cascading Style Sheets«.

- entweder direkt im Quellcode der HTML-Datei:

```
<!-- Meist erfolgt die Festlegung im head des HTML-Dokuments -->
<style type="text/css">
body { color: red;}
</style>

<!-- Oder im Tag direkt -->
<h1 style="text-align: center;">
Mittiger Text in h1 Groesse
</h1>
```

- oder in ausgelagerten Dateien mit der Endung `.css`, auf die dann im Header der HTML-Datei referenziert wird:

```
<head>
...
<link rel="stylesheet" type="text/css" ref="test.css" />
</head>
```

CSS folgt, wie man in den Beispielen oben sehen kann, immer dem gleichen Aufbau:

```
selektor { eigenschaft:wert; }
```

CSS galt lange Zeit als nicht besonders relevant in Bezug auf die Sicherheit von Webapplikationen. Allerdings haben vor allem Neuerungen und Funktionserweiterungen dazu geführt, dass sich der Einfluss von CSS auf Applikationen merklich vergrößert hat, zudem gab es immer wieder Angriffe, die CSS gezielt nutzten, um die verwundbaren Webseiten zu manipulieren. Insoweit spielt CSS mittlerweile eine wichtige Rolle, auch da sich der HTML-Code und dessen Interpretation durch CSS stark verändern können. Wir werden vor allem im Kapitel über UI-Redressing sehen, dass durch CSS einige Angriffe ermöglicht werden (siehe dazu Kapitel 11, »Fehler! Verweisquelle konnte nicht gefunden werden.«).

CSS wird vom W3C (World Wide Web Consortium) standardisiert [13] und wurde zuletzt unter der Version 2.1 veröffentlicht [14], die meist als CSS 3 bezeichnet wird.

JavaScript

HTML und CSS wurden später durch JavaScript ergänzt, wodurch nicht nur ein Strukturieren und Gestalten, sondern auch ein clientseitiges Verändern, Validieren und Generieren von Inhalten ermöglicht wurde. JavaScript wird wie HTML und CSS erst innerhalb des Webbrowsers interpretiert und ermöglicht eine Vielzahl von prakti-

schen Funktionen, die zuvor nicht so einfach umsetzbar waren, etwa »Bewegung« innerhalb von Webseiten.

JavaScript ist eine clientseitige Skriptsprache und wurde 1997 unter der heute kaum bekannten Bezeichnung »ECMA-Script« (European Computer Manufactures Association Script) standardisiert. Das heutige JavaScript müsste also korrekterweise ECMA-Script lauten. JavaScript wird aktuell von allen modernen Webbrowsern unterstützt und findet in vielen Webapplikationen Verwendung, etwa um Daten während der Eingabe zu validieren oder Suchbegriffe vorzuschlagen.

Wie wir in dem Abschnitt über Formulare und Input-Tags gesehen haben, gab es vor HTML5 keine wirkliche Möglichkeit, Eingaben verbindlicher vorzuschreiben, daher nahm man die Validierung der eingegebenen Werte häufig mittels JavaScript vor.

JavaScript hat sich als Standard für clientseitige Skriptsprachen etabliert und andere Skriptsprachen weitestgehend verdrängt. Es kann, wie CSS auch, direkt im Quellcode implementiert sein:

```
001 <script language="javascript" type="text/javascript">
002 <!-- Hier steht JavaScript-Code -->
003 </script>
```

Seit HTML5 genügt auch folgende Einbindung:

```
001 <script>
002 <!-- Hier steht JavaScript-Code -->
003 </script>
```

Die Einbindung von JavaScript als externe Datei in eine HTML-Seite (HTML-Header) ist ebenfalls möglich und sieht folgendermaßen aus:

```
001 ...
002 <head>
003 <script language="javascript" type="text/javascript" scr="datei.js">
    </script>
004 <!-- Folgende Einbindung ist auch möglich -->
005 <script src="datei.js"></script>
006 </head>
007 ...
```

JavaScript gleicht sehr stark klassischen Programmiersprachen und verfügt über bedingte Anweisungen, Schleifen aller Art sowie Variablen, und es lassen sich Funktionen definieren.

Folgendes Beispiel ist ein typisches Programm in JavaScript:

```
001 <script>
002 var x = multiplikation(3, 4);
003
004 function multiplikation(a, b) {
005    return a * b;
006 }
007 </script>
```

Das Ergebnis des Funktionsaufrufs von `multiplikation` wäre in diesem Fall 12 (3*4).

Der entscheidende Vorteil von JavaScript ist, dass Inhalte dynamisch, unmittelbar und ohne erneute Anfrage an den Server verarbeitet und angezeigt werden können. Es besteht nicht mehr die Notwendigkeit, eine Webseite neu zu laden, wenn geringfügige Änderungen vollzogen wurden, stattdessen können mit JavaScript Inhalte direkt validiert werden.

Es ist also mit JavaScript zum Beispiel möglich, Nutzern direkt ein Feedback zu geben, während diese noch ihre Daten eingeben. Dadurch kann verhindert werden, dass unzureichende Daten an den Server geschickt werden und dieser mit einer Error-Ausgabe antworten muss. Durch die clientseitige Eingabenvalidierung von HTML5 wurde ein Teil dieser Problemlösung übernommen. Allerdings werden viele Meldungen und Funktionen auf Webseiten nach wie vor mit JavaScript implementiert, denn HTML ist in der ursprünglichen Idee ausschließlich eine Auszeichnungssprache gewesen, JavaScript hingegen war von Anfang an für komplexere Funktionen wie Validierungen vorgesehen.

Vor der Einführung von JavaScript wäre für jede fehlerhafte Eingabe jeweils eine Anfrage an den Server und eine Antwort von diesem vonnöten gewesen, in der darum gebeten werden müsste, die Eingabe zu überprüfen. Mit JavaScript konnte das direkt bei Eingabe der Daten geschehen.

Zudem kann mit JavaScript auf Cookies zugegriffen werden — sowohl lesend als auch schreibend — über das sogenannte DOM (Document Object Model), das wir im Kapitel über Webbrowser genauer betrachten werden (siehe Kapitel 3.4.1, »Document Object Model (DOM)«).

Eine Vielzahl von Webbrowsern hat die Befugnisse von JavaScript stark eingeschränkt, gerade weil sich damit tatsächlich massive Veränderungen vornehmen lassen. Nahezu alle Browser verfügen mittlerweile über Pop-up-Blocker, die das Erscheinen lästiger Fenster oder aufdringlicher Werbung unterbinden sollen. Die meisten Webbrowser bieten zudem die Möglichkeit, JavaScript komplett zu deaktivieren, ein

Großteil der Webseiten im Web dürfte dann allerdings nicht mehr benutzbar sein, da JavaScript in nahezu allen modernen Webanwendungen eingesetzt wird.

Ajax

Ajax steht für »Asynchronous JavaScript and XML« und ist eine Kombination aus den uns bereits bekannten Technologien XML und JavaScript.[5] Zentraler Vorteil von Ajax ist, dass eine clientseitige Verarbeitungsebene, die sogenannte Ajax-Engine, geschaffen wird. Dadurch können HTTP-Anfragen vorgenommen werden, während die HTML-Seite angezeigt wird. Im Kapitel über Webbrowser werden wir sehen, dass diese Ajax-Engine auf das DOM zurückgreift und dadurch Inhalte unmittelbar beeinflussen kann.

Klassischerweise muss eine Seite im HTTP-Protokoll nach einer Änderung immer neu geladen werden, jedoch nicht, wenn sie eine Ajax-Technologie nutzt. Der Webbrowser erhält nach einer Anfrage an eine Webapplikation, die Ajax nutzt, zunächst eine gewöhnliche Antwort mit XML- oder HTML- und JavaScript-Dateien. Diese werden im Webbrowser durch die Ajax-Engine verarbeitet, und es werden HTML- und CSS-Dateien lokal generiert und angezeigt. Wenn Neuerungen an den Daten vorgenommen werden, muss nicht die gesamte Seite neu geladen werden, stattdessen werden durch JavaScript-Aufrufe lediglich einzelne Anfragen durchgeführt. Dadurch wird vermieden, dass eine Vielzahl von Daten mehrmals übermittelt werden muss, und die Benutzung der Webapplikation wird vor allem aufgrund kürzerer Ladezeiten angenehmer.

Ein simples Codebeispiel soll zeigen, wie Ajax in etwa funktioniert (allerdings ist das komplette Verständnis dieses Codes wohl erst dann gegeben, wenn man verstanden hat, wie das DOM aufgebaut ist):

```
001 <html>
002 <head>
003 <script>
004 function lade()
005 {
006 var xmlhttp;
007 if (window.XMLHttpRequest)
008    {
009    xmlhttp=new XMLHttpRequest();
010    }
011 xmlhttp.onreadystatechange=function()
```

[5] XML ist für die Nutzung von Ajax — anders als es der Name vielleicht suggeriert — nicht zwingend notwendig, es können beispielsweise auch mittels JSON Daten verarbeitet werden (siehe nächstes Unterkapitel).

```
012     {
013     if (xmlhttp.readyState==4 && xmlhttp.status==200)
014        {
015        document.getElementById("Bereich").innerHTML=xmlhttp.
    responseText;
016        }
017     }
018  xmlhttp.open("GET","ajax.txt",true);
019  xmlhttp.send();
020  }
021  </script>
022  </head>
023
024  <body>
025  <div id="Bereich"><h3>Hier steht austauschbarer Text</h3></div>
026  <button type="button" onclick="lade()">Wechsle Inhalt</button>
027  </body>
028  </html>
```

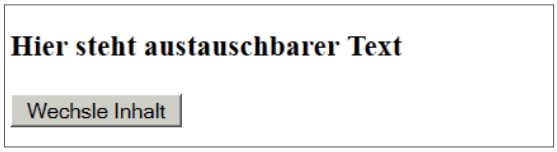

Bild 5: Darstellung des oben angeführten Codes.

Es ist ausschließlich der `body`-Bereich zu sehen. Ein Klick auf den Button führt allerdings die im HTML-Kopf (Head) definierte Funktion `lade()` aus und sorgt dafür, dass der Inhalt aus dem `div`-Bereich `Bereich` durch den Inhalt der `ajax.txt` ersetzt wird.

Ajax nutzt die XMLHttpRequest-Technologie (XHR), eine Programmierschnittstelle in JavaScript, die es ermöglicht, im Hintergrund Anfragen zu versenden und dynamisch Inhalte von entsprechenden Webservern abzurufen und im DOM einzufügen.

Dabei werden alle gängigen HTTP-Methoden unterstützt, und ein eigenes Schema für die Beschreibung von Zuständen, die sogenannten `readyState`-Zustände, wird verwendet. Zusammen mit den Statuscodes, die uns bereits aus dem HTTP-Kapitel bekannt sind, lassen sich so einfache Abfragen erzeugen. Diese Abfragen benötigen, wie bereits mehrfach erwähnt, kein komplettes Neuladen der Webseite, sondern nur das Laden einzelner Bestandteile.

Eigenschaft (Wert)	Beschreibung
onreadystate-change	Speichert eine Funktion bzw. ihren Namen, um sie beim Wechsel des readyState-Werts aufzurufen.
readyState	Beinhaltet den Status des XMLHttpRequests: 0: Anfrage nicht initialisiert 1: Serververbindung hergestellt 2: Anfrage erhalten 3: Verarbeite Anfrage 4: Anfrage abgeschlossen, Antwort erhalten
status	200: OK 404: Inhalt nicht gefunden

Ajax hat mit Blick auf die Sicherheit dazu geführt, dass eine Menge neuer Angriffsvektoren entstanden ist (dazu an anderer Stelle mehr, wenn Ihnen genauer bekannt ist, wie ein Webbrowser und das DOM funktionieren).

JSON

JSON (JavaScript Object Notation) ist ein Datenformat, das mittels JavaScript leicht erzeugt und angesprochen werden kann. JSON wird vor allem in der zuvor beschriebenen Ajax-Technologie genutzt und bietet eine Alternative zu XML.

Einige Webapplikationen verarbeiten Anfragen mittels XHR nicht, indem sie mit einer XML-Datei antworten, sondern sie nutzen in Antworten JSON. Moderne Browser sind unabhängig von der Art der Antwort in der Lage, diese zu verarbeiten.

JSON sieht folgendermaßen aus:

```
001 {
002    "id": "253",
003    "name": "Test Nutzer",
004    "geschlecht": "männlich",
005    "sprache": "deutsch",
006    "nutzername": "test"
007 }
```

Ein Browser könnte mittels JavaScript die relevanten Daten verarbeiten. Dazu werden diese zunächst herausfiltert und dann an den entsprechenden Stellen eingefügt.

VBScript

Nachdem wir viele clientseitige Sprachen besprochen haben, möchten wir unseren Blick abschließend auf VBScript (Visual Basic Script) richten. Dabei handelt es sich um einen historischen Versuch von Microsoft, der Verbreitung und Popularität von JavaScript Einhalt zu gebieten und im Bereich der clientseitigen Skriptsprachen einen Standard zu etablieren — was jedoch misslang.

VBScript wurde 1996, also vor rund 20 Jahren, eingeführt und wird bis heute vom Internet Explorer unterstützt — Microsoft hat jedoch bereits angekündigt, dass im neuen Spartan Browser (Nachfolger des Internet Explorers) die Unterstützung weitgehend eingestellt wird [15].[6]

Unter dem Aspekt der Websicherheit hat VBScript bislang nur eine begrenzte Relevanz gehabt, da die Codeausführung primär innerhalb des Internet Explorers funktionierte. Allerdings wurden viele bekannte Computerwürmer, wie etwa der sogenannte »I-LOVE-YOU-Virus«, in VBScript geschrieben, da jedes Microsoft-Betriebssystem ab Windows 98 und Programme wie Outlook allesamt ohne zusätzliche Installation VBScript interpretieren konnten.

Weitere clientseitige Technologien

Neben den bereits vorgestellten clientseitigen Sprachen und Technologien, die in allen Webbrowsern vorhanden sind, gibt es weitere clientseitige Technologien, die überwiegend über Browsererweiterungen hinzugefügt und genutzt werden können.

Dazu gehören zum Beispiel Flash (ActionScript), ActiveX, Java-Applets oder PDF-Dateien. Wie bereits in der Einleitung erwähnt, werden wir diese Technologien nicht umfassend behandeln, sondern lediglich später bei entsprechenden Schwachstellen einzelne Angriffsvektoren aufzeigen.

Serverseitige Sprachen

Wie wir im Kapitel über die Evolution des Webs bereits gelernt haben, waren die Webseiten der ersten Stunde allesamt statisch. Server hatten HTML-Dateien und andere statische Dateien gespeichert, die auf Anfrage von Clients übertragen wurden.

Heutzutage hat sich viel verändert, hinter den Servern stehen zumeist Subsysteme wie Datenbanken, die Applikationen mit Datensätzen ausstatten. Zwar ist immer noch ein großer Teil der Inhalte statisch (beispielsweise meist das Design — das

[6] Anders als in vielen Medien berichtet, ist VBScript damit allerdings noch nicht ganz »tot«, mittels `<msxsl:script language="VBScript">` kann es nach wie vor verwendet werden.

CSS — einer Webseite), ein anderer Teil wird ABER dynamisch erzeugt (häufig nutzerspezifische Daten).

Um die dynamischen Inhalte zu erstellen, zu verarbeiten und an den Nutzer auszugeben, benötigt der Server Programmiersprachen bzw. Webframeworks, die die Verarbeitung vornehmen. Wir wollen im Folgenden die wichtigsten serverseitigen Sprachen kurz betrachten, um zu verstehen, wie das Web arbeitet, damit wir in den Kapiteln danach mögliche Attacken ausmachen und Codebeispiele verwenden können.

Laut einer repräsentativen Umfrage des »W3-Techs — Web Technology Surveys« [16] verteilt sich die Nutzung der Top 6 serverseitigen Skriptsprachen folgendermaßen:

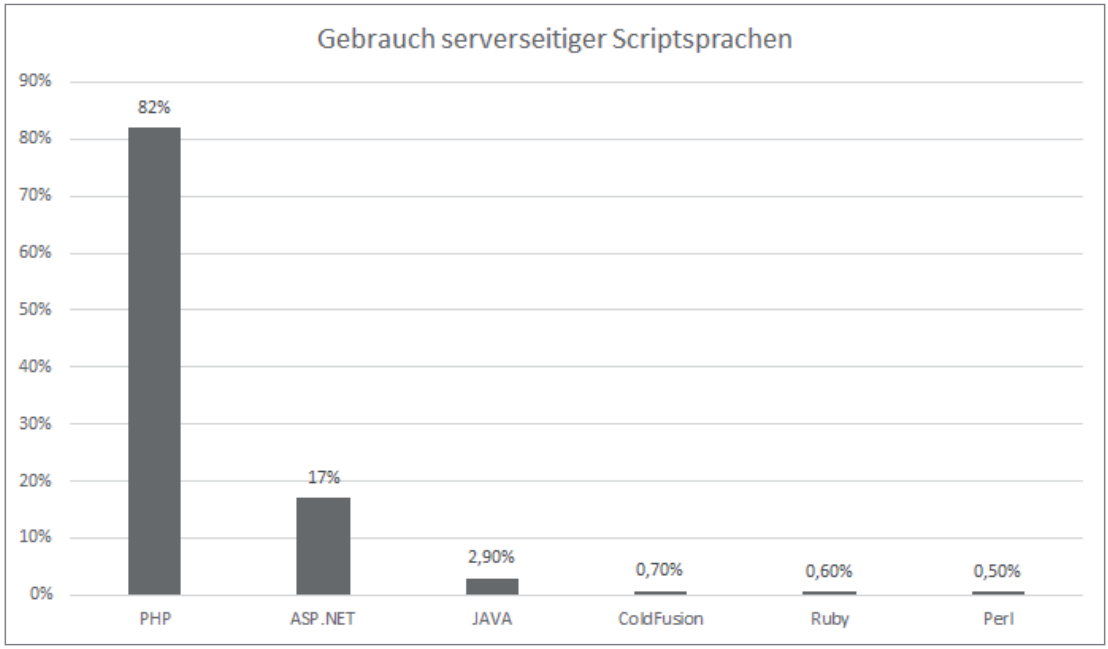

Bild 6: Anteilige Darstellung der Programmiersprachen im Web.

PHP

PHP (PHP Hypertext Preprocessor)[7] ist die mit Abstand am häufigsten verwendete serverseitige Skriptsprache im Web. Sie wurde im Jahr 1995 von Rasmus Lerdorf entwickelt und war zunächst eine Ansammlung von C-Programmen und Perl-Skripten.

Im Jahr 1997 wurde die Codebasis von Andi Gutmans und Zeev Suraski mit Einverständnis von Rasmus Lerdorf komplett neu geschrieben und 1998 als PHP 3.0.0 veröffentlicht. Da in der Boomphase des Webs Ende der 1990er-Jahre (siehe Kapitel 2.2, »Fehler! Verweisquelle konnte nicht gefunden werden.s«) ein erhöhter Bedarf an dynamischen Webseiten entstand und PHP einen soliden und vor allem plattformunabhängigen und kostenlosen Standard bot, erfreute sich das PHP-Projekt großen Zulaufs.

Nach und nach wurden immer wieder Ausbesserungen vorgenommen, beispielsweise wurde mit PHP 4 die Verwendung globaler Variablen verbessert und eine optimierte Ausführungsgeschwindigkeit ermöglicht. Nach PHP 4 wurde mit PHP 5 im Jahr 2004 ein lang anhaltender Standard geschaffen, denn die aktuelle Version ist PHP 5.6.X. PHP 5 hält sich also schon über zehn Jahre.

Zwischenzeitlich sollte eigentlich schon PHP 6 veröffentlicht werden, davon wurde allerdings abgesehen, offensichtlich, da es Probleme bei der Implementierung des neuen Unicodes gab [17]. Die meisten anderen Änderungen wurden jedoch in den entsprechenden Unterversionen von PHP 5 berücksichtigt. Zuletzt sorgte das Nichtveröffentlichen von PHP 6 immer wieder für große Debatten innerhalb der PHP-Community, wie man eine neue Version nennen solle. Letztlich wurde sich dafür entschieden, die neueste PHP-Version mit dem Titel PHP 7 zu versehen [18].

Die finale Version von PHP 7 wurde im Dezember 2015 veröffentlicht. Die Version erscheint mit einer neuen Hashtabelle, die eine Verkürzung der Ausführungszeit ermöglichen soll.

Momentan werden noch die Versionen PHP 5.4, PHP 5.5 und als neueste Version PHP 5.6 unterstützt.

[7] Ursprünglich wurde PHP unter dem Akronym »Personal Home Page Tools« veröffentlicht.

Allerdings sind bereits Zeiträume geplant, um den Support für die bestehenden Versionen einzustellen:

Version	Veröffentlichung	Ende des aktiven Supports	Ende des sicherheitsbezogenen Supports
5.4	01.03.2012	14.09.2014	14.09.2015
5.5	20.06.2013	20.06.2015	20.06.2016
5.6	28.08.2014	28.08.2016	28.08.2017

PHP bietet, wie oben bereits beschrieben, eine Reihe von entscheidenden Pluspunkten: Neben den bereits benannten Vorteilen der Plattformunabhängigkeit und dem kostenlosen Einsatz (dank eigener PHP-Lizenz) kommen die einfache Handhabung (Syntax) sowie die Möglichkeit zur Verbindung mit einem Datenbanksubsystem (etwa mit MySQL, siehe Abschnitt »SQL«) hinzu.

PHP eignet sich aufgrund seiner simplen Grundsätze auch als Programmiersprache für Anfänger und gilt mitsamt einiger Frameworks als äußerst programmierungsfreundlich.

Viele bekannte Projekte (vor allem aus dem Open-Source-Bereich) basieren auf einem PHP-Quelltext, die bekanntesten Vertreter sind nachfolgend aufgeführt:

- **Content-Management-Systeme** (**CMS**): TYPO3, WordPress, Joomla
- **E-Commerce-Software**: Magneto, OpenCart, osCommerce
- **Forensoftware:** MyBB, phpBB
- **Fotogalerien:** Gallery, phpAlbum, UberGallery
- **Webmail-Software:** Roundcube, SquirrelMail
- **Webtracking-Software:** Open Web Analytics, Piwik
- **Wiki-Systeme:** MediaWiki, TWiki, DokuWiki, FlexWiki

Neben diesen bekannten PHP-Programmen gibt es natürlich viele weitere, zudem ist jeder selbst in der Lage, PHP-Programme zu schreiben und auszuführen. Viele Firmen haben eigene PHP-Applikationen in Gebrauch, die Einsatzmöglichkeiten sind dabei nahezu unbegrenzt (siehe Kapitel 2.3, »Fehler! Verweisquelle konnte nicht gefunden werden.«).

Werfen wir einmal einen Blick in den Quellcode einer simplen PHP-Anwendung:

```
001 <?php
002 $cookie_name = "nutzer";
003 $cookie_wert = "Test Nutzer";
004 setcookie ($cookie_name, $cookie_wert, time() + (3600), "/"); //3600 =
    1 Stunde
005 ?>
006 <html>
007 <body>
008
009 <?php
010 if(!isset($_COOKIE[$cookie_name]))
011    {
012    echo "Ein Cookie mit dem Namen ," . $cookie_name . "‘ ist nicht
    gesetzt!";
013    }
014    else
015    {
016    echo "Cookie ‘" . $cookie_name . "‘ ist gesetzt!<br>";
017    echo "Der Wert ist: " . $_COOKIE[$cookie_name];
018    }
019 ?>
020 <p>Zum Anzeigen des Cookies bitte die Seite neu laden!</p>
021 </body>
022 </html>
```

Das PHP-Skript setzt ein Cookie mit der PHP-Funktion `setcookie()`, das für eine Stunde gültig ist. Der Name des Cookies ist `nutzer`, und es enthält den Wert `Test Nutzer`. Die `echo`-Befehle am Ende des Skripts geben die zuvor gesetzten Werte der Cookies auf der Webseite aus.

Wie dem Beispiel zu entnehmen ist, kann man eine Verschachtelung von HTML und PHP ohne Probleme vornehmen. PHP-Dateien können auch CSS oder JavaScript enthalten.

Die Teile, in denen PHP verwendet wird, werden jeweils mit `<?php` begonnen und enden mit `?>`. Damit der PHP-Code auch entsprechend ausgeführt wird, ist es wichtig, dass die Datei auf `.php` endet. Mit geschweiften Klammern `{}` werden die einzelnen Sinnabschnitte voneinander abgegrenzt.

PHP bietet den vollen Umfang einer Programmiersprache (Deklaration von verschiedenen Datentypen, Erstellung von Funktionen und Klassen) und wird im Web größtenteils eingesetzt, um:

- dynamische Inhalte zu generieren,
- Dateien zu öffnen, zu bearbeiten und zu speichern,
- Cookies zu setzen oder zu empfangen sowie
- Zugriffe auf Datenbanken durchzuführen.

Aufgrund der großen Verbreitung von PHP werden wir immer wieder Codebeispiele in PHP betrachten.

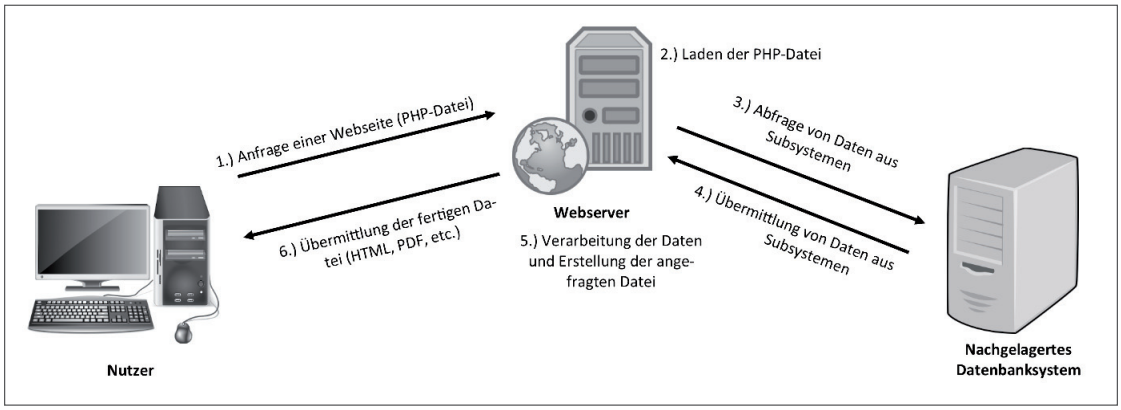

Bild 7: Aktionen bei Aufruf einer PHP-Datei.

Grundsätzliche Sicherheitshinweise zu PHP können unter *php.net/manual/de/security.php* eingesehen werden.

Neben den vielen oben genannten Systemen wird PHP auch bei einigen bekannten Diensten im Web, beispielsweise bei Facebook, Apple, Vimeo und Flickr, eingesetzt.

ASP.NET

ASP.NET (Active Server Pages .NET) ist eine Methode zum Erstellen von Webseiten, die von Microsoft entwickelt wurde und die Skriptsprache ASP (Active Server Pages) 2002 ablöste. Unter Einsatz von .NET-Sprachen, etwa C# oder VB.NET, können Webapplikationen erstellt werden. ASP.NET ist jedoch nicht als reine Programmiersprache zu betrachten, da eine Vielzahl von Technologien zum Einsatz kommt.

Der Vorteil dabei ist, dass keine komplexe Programmierung notwendig ist, vielmehr kann man das Framework und bereits vorhandene Vorlagen nutzen. So gibt es in ASP.NET beispielsweise schon umfangreiche Möglichkeiten zur Authentifizierung, Autorisierung und zum Steuern von Benutzerrechten. Weiterhin ist eine Verbindung mit einem Active Directory und dem entsprechenden Service zur Authentifizierung von

Benutzern (Active Directory Federation Services, ADFS) ohne Weiteres möglich. Eine Verbindung zu Subsystemen wie Datenbanken ist ebenfalls leicht möglich. Die aktuellste Version ist ASP.NET 5, die 2015 veröffentlicht wurde.

Ein Beispiel in ASP.NET (mittels VB.NET) sieht folgendermaßen aus:

```
@Code
Dim wochentag=DateTime.Now.DayOfWeek
Dim tag=wochentag.ToString()
Dim nachricht=""
End Code
<html>
<body>
@Select Case tag
Case "Montag"
   nachricht="Heute ist der erste Tag der Woche. "
Case "Freitag"
   nachricht="Das Wochenende naht :)!"
Case Else
   nachricht="Heute ist " & tag
End Select
<p>@nachricht</p>
</body>
</html>
```

Der Code von oben ist in VB.NET geschrieben und ermittelt den aktuellen Wochentag, um in einer `Select Case`-Anweisung eine entsprechende Nachricht zu erzeugen. Diese wird als Paragraf (`<p></p>`) innerhalb des HTML-Codes ausgegeben.

Das gleiche Beispiel sieht in C# etwa folgendermaßen aus:

```
@{
var wochentag=DateTme.Now.DayOfWeek;
var tag=wochentag.ToString()
var nachricht="";
}
<html>
<body>
@switch(tag)
{
Case "Montag"
   Nachricht="Heute ist der erste Tag der Woche.";
   break;
case "Freitag":
```

```
014     message="Das Wochenende naht :)!";
015     break;
016  default:
017     nachricht="Heute ist " + tag;
018     break;
019  }
020  <p>@nachricht</p>
021  </body>
022  </html>
```

Es kommt die für C-Programme charakteristische `switch`-Anweisung zum Einsatz. Des Weiteren wird die Deklaration der Variablen nicht mittels `Dim` vorgenommen, `var` genügt hier. Es gibt weitere Unterschiede, so werden die Teilbereiche in VB.NET mit einer Art Code-Tag umschlossen (Beginn: `@Code`, Ende: `End Code`), während bei der Verwendung von C# geschweifte Klammern `{}` zum Einsatz kommen.

Sicherheitshinweise zu ASP.NET sind unter *msdn.microsoft.com/en-us/library/330a99hc%28v=vs.140%29.aspx* zu finden.

ASP.NET ist zumeist dort im Einsatz, wo Produkte von Microsoft genutzt werden, insoweit nutzen viele Firmen es beispielsweise für das Intranet (Sharepoint). Zudem basieren die Webanwendungen Bing und MSN darauf.

Insgesamt soll das als erster Eindruck genügen, wir werden sicherlich an geeigneter Stelle erneut auf ASP.NET zu sprechen kommen.

Java Platform EE

Weniger als 3 % der Webapplikationen nutzen Java Platform EE, um dynamische Inhalte zu erzeugen — aus diesem Grund nur ein kurzer und auch eher oberflächlicher Einblick in diese Technologie, überwiegend in sogenannte Servlets.

Java EE steht für »Java Enterprise Edition« (ehemals J2EE) und basiert auf Java. Es wurde ab 1999 vom Unternehmen Sun Microsystems entwickelt, das 2010 von Oracle aufgekauft wurde.

Grundsätzlich lässt sich bei Webapplikationen in Java Platform EE folgender Aufbau erkennen:

```
001  Projekt/
002     -- *.html, *xml, *.jpg, *.pdf *.jsp, …
003     -- WEB-INF/
004        -- web.xml
005        -- classes/
006     -- lib/
```

Das oberste Verzeichnis (`Projekt`) wird auch »Document Root« genannt, enthält alle weiteren Ordner und kann beliebig benannt werden.

Innerhalb von Java Platform EE ist eine klarere Trennung zwischen Inhalten, die für den Nutzer vorgehalten werden, und jenen, die für das interne Programm und die Verarbeitung der Inhalte benötigt werden, möglich.

Der Ordner `WEB-INF` enthält die Datei `web.xml`, diese wird als »Deployment Descriptor« bezeichnet und regelt maßgeblich, welche Klassen und Ressourcen genutzt werden sollen, um HTTP-Anfragen zu verarbeiten.

Eine typische `web.xml` könnte folgendermaßen aussehen:

```
001 <web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
002     <servlet>
003         <servlet-name>projekt</servlet-name>
004         <servlet-class>webseite.server.projektServlet</servlet-class>
005     </servlet>
006     <servlet-mapping>
007         <servlet-name>projekt</servlet-name>
008         <url-pattern>/*</url-pattern>
009     </servlet-mapping>
010 </web-app>
```

Wie man erkennen kann, handelt es sich um eine XML-Datei, in der ein sogenanntes Servlet erstellt und gemappt wurde. Bei Servlets handelt es sich um einfache Java-Klassen (Servletklassen). Diese Klassen erben wiederum von Standardklassen oder können an Datenbanken geknüpft sein, um ihre Funktionen voll zu erfüllen. Die Verarbeitung und Beantwortung der HTTP-Anfragen findet durch die über die `web.xml` ermittelten Servlets statt, und der Nutzer erhält durch das entsprechende Servlet auch eine HTTP-Antwort.

In der oben angeführten `web.xml` werden also sämtliche URLs, die angefragt werden, mit dem entsprechenden Servlet `webseite.server.projekt` bedient.

Was dann im Einzelnen passiert, ist von dem angesprochenen Servlet abhängig.

Nehmen wir einmal an, dass es folgendermaßen aussieht:

```
001 import java.io.IOException;
002 import java.io.PrintWriter;
003 import javax.servlet.ServletException;
004
005 import javax.servlet.http.HttpServlet;
006 import javax.servlet.http.HttpServletRequest;
```

```
007 import javax.servlet.http.HttpServletResponse;
008
009 public class projekt extends HttpServlet {
010    public void doGET(HttpServletRequest request, HettpServletResponse
    throws ServletException, IOException {
011       response.setContentType("text/html");
012       PrintWriter out = response.getWriter();
013       out.println("<html>");
014       out.println("<head><title>Hier steht ein Titel</title></head>");
015       out.println("<body>");
016       out.println("<h1>Hier steht nun ganz viel HTML</h1>");
017       /* funktion {
018       hier können Java-Funktionen platziert werden
019       }
020       */
021       out.println("</body>");
022       out.println("</html>");
023       out.close();
024    }
025 }
```

Der Programmcode sorgt in diesem Fall für eine HTML-Datei, die folgendermaßen aussieht und an den Nutzer geschickt wird:

```
001 <html>
002 <head><title>Hier steht ein Titel</title></head>
003 <body>
004 <h1>Hier steht nun ganz viel HTML</h1>
005 </body>
006 </html>
```

Man könnte den Programmcode oben auch mit entsprechenden Java-Funktionen spicken und somit extrem komplexe Sachverhalte abbilden. Dieser Code würde auf dem Server ausgeführt und kann anschließend dem Nutzer übergeben werden. Entscheidender Vorteil von Java ist an dieser Stelle die Fehlerbehandlung, die mittels des sogenannten Exception-Handlings leichterfällt als beispielsweise in PHP. Java EE besitzt eine Vielzahl von Klassen, die, zugeschnitten auf die Anforderungen eines Programms, mithilfe der Importfunktion genutzt werden können. Aus diesem Grund wird Java EE überwiegend im Businessbereich eingesetzt. Java EE gilt als nicht besonders eingängig für Anfänger.

Bekannte Webseiten, die Java im Einsatz haben, sind beispielsweise Amazon, eBay und LinkedIn.

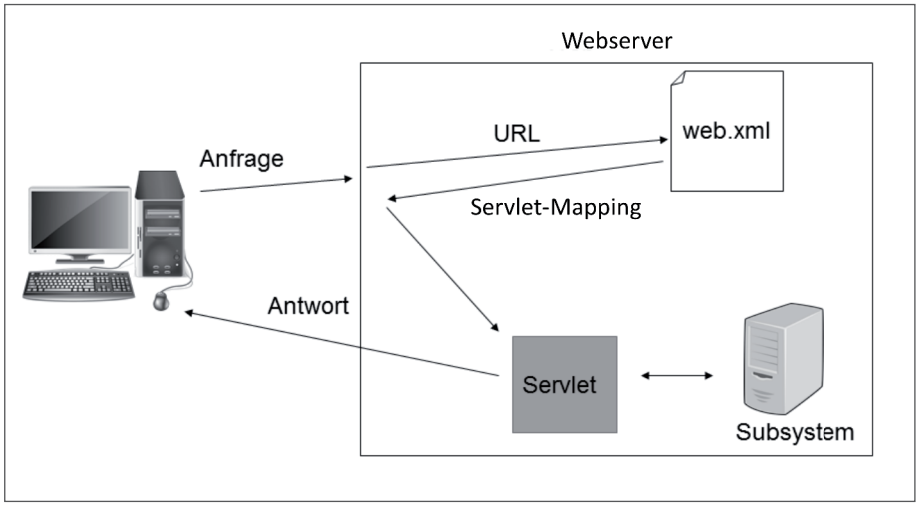

Bild 8: Anfrage an einen Webserver mit Java Platform EE.

Java Platform EE ist umfassend dokumentiert, Informationen zur Sicherheit lassen sich beispielsweise unter *docs.oracle.com/javaee/6/tutorial/doc/gijrp.html* einsehen.

ColdFusion

ColdFusion wurde 1995 entwickelt. Später wurde die Firma Allaire, die maßgeblich an der Entwicklung beteiligt war, von Macromedia gekauft und diese wiederum von Adobe. Bis heute wird ColdFusion von Adobe entwickelt und bietet wie die Sprachen zuvor die Möglichkeit, Inhalte auf Webseiten dynamisch zu gestalten. Dabei ist ColdFusion sogar in der Lage, mit anderen Programmiersprachen wie .NET oder Java zu interagieren. Die aktuelle Version ist ColdFusion 11, das neueste Update erschien Ende 2014.

Besonders wichtig ist in ColdFusion die Datei `application.cfm`. In dieser können Werte oder Funktionen festgelegt werden, sie wird vor allen anderen Teilen der Applikation ausgeführt. Jedes Mal, wenn man also eine ColdFusion-Webapplikation aufruft, etwa `index.cfm`, wird zunächst geprüft, ob eine `application.cfm` vorliegt. Ist das der Fall, wird sie inkludiert, liegt sie nicht vor, wird direkt die angefragte Datei ausgeführt. Das Inkludieren kann auch mit

```
<cfinclud>test.cfm</cfinclude>
```

erreicht werden.

Das Verhalten gleicht der Include-Funktion in PHP:

```
001 <?php
002 include ‚test.php‘;
003 ?>
```

Die `application.cfm` eignet sich also gut, um grundsätzliche Funktionen (etwa das Session-Management) zu implementieren.

Ansonsten ist wichtig zu erwähnen, dass ColdFusion nicht nur die Funktion einer Programmiersprache bietet, sondern auch einen Applikationsserver. Damit ist ColdFusion eine typische Middlewareapplikation, mit der man auch Datenbanken und andere Webservices ansprechen kann.

Sicherheitsmeldungen zu ColdFusion werden auf *www.adobe.com/devnet/coldfusion/security.html* publiziert.

Ruby (on Rails)

Ruby on Rails (ehemals RoR oder einfach kurz »Rails«) erfreut sich in den letzten Jahren immer größerer Beliebtheit. RoR wurde erstmals 2004 mit der Version 1.0 veröffentlicht und ist somit einer der moderneren Standards. Es handelt sich um ein Webframework, das auf der Programmiersprache Ruby aufbaut. Die Entwickler haben sich zum Ziel gesetzt, eine besonders einfache Art des Programmierens zu ermöglichen, und das Prinzip »Don't Repeat Yourself« (DRY)« angewendet, dem sie bis heute treu geblieben sind.

Rails bietet in der Tat einige Vorteile. Es ist sehr schnell, und man kann als Entwickler, wenn man sich an das Programmierungsprinzip hält und zum Beispiel Namenskonventionen intensiv nutzt, gegenüber Sprachen wie PHP einiges an Codezeilen sparen.

Da Rails nicht nur eine reine Programmierungssprache, sondern auch ein Webframework ist, wird bei der Erstellung einer neuen Seite eine Vielzahl von Ordnern erzeugt, etwa zum Logging oder zur Konfiguration.

Zu aktuellen Sicherheitsmeldungen zu Ruby on Rails empfiehlt sich *www.ruby-lang.org/en/security/*.

Bekannte Webseiten, die auf Ruby on Rails basieren, sind zum Beispiel Airbnb, GitHub, Shopify und Twitter.

Perl

Perl ist mit Abstand eine der ältesten Programmiersprachen, die bis heute im Web verwendet werden, sie wurde bereits 1987 von Larry Wall entwickelt und erinnert überwiegend an die Programmierung in C. Zunächst wurde Perl nicht für die Pro-

grammierung im Web konzeptioniert, allerdings kam ab Version 5 (1994) auch die Möglichkeit hinzu, es für die Webprogrammierung zu nutzen. Damit war Perl eine der ersten Programmiersprachen des Webs überhaupt. Bei Perl handelt es sich also um einen alten Standard, der bis heute überdauert hat — und zwischenzeitlich sogar einige Neuentwicklungen wie zum Beispiel PHP stark beeinflusst hat.

Damit Perl benutzt werden kann, wird meist auf CGI (Common Gateway Interface) zurückgegriffen, einen Standard zum Datenaustausch, der 1993 entwickelt wurde.[8]

Ein klassisches Programm, um einen Vor- und einen Nachnamen mittels `GET`-Parameter einzulesen und auszugeben, sieht etwa so aus:

```
001 #!/usr/bin/perl
002     # Erstellt Parameter
003     local ($buffer, @pairs, $pair, $name, $value, %FORM);
004     $ENV{,REQUEST_METHOD‘} =~ tr/a-z/A-Z/;
005     # Wenn eine Anfrage mittels GET-Methode kommt
006     if ($ENV{,REQUEST_METHOD‘} eq "GET")
007     {
008    # Übergebe GET-Parameter aus der Umgebungsvariablen
009    $buffer = $ENV{,QUERY_STRING‘};
010     }
011     # Teilt die Werte entsprechend auf
012     @pairs = split(/&/, $buffer);
013     foreach $pair (@pairs)
014     {
015    ($name, $value) = split(/=/, $pair);
016    $value =~ tr/+/ /;
017    $value =~ s/%(..)/pack("C", hex($1))/eg;
018    $FORM{$name} = $value;
019     }
020     $first_name = $FORM{vor_name};
021     $last_name  = $FORM{nach_name};
022
023 print "Content-type:text/html\r\n\r\n";
024 print "<html>";
025 print "<head>";
026 print "<title>Programm zur Namensausgabe</title>";
027 print "</head>";
028 print "<body>";
029 print "<h1>Hallo $vor_name $nach_name – alles Gute!</h1>";
```

[8] Mittlerweile ist die Ausführung von Perl beispielsweise auch über Middleware wie »PSGI/Plack« (*plackperl.org*) möglich.

```
030 print "</body>";
031 print "</html>";
032
033 1;
```

Wenn wir als Vornamen »Peter« übergeben und als Nachnamen »Schmidt«, wird folgende Webseite erzeugt:

```
001 <html>
002 <head><title>Programm zur Namensausgabe</title></head>
003 <body>
004 <h1>Hallo Peter Schmidt – alles Gute!</h1>
005 </body>
006 </html>
```

Wie Sie merken, orientiert sich Perl also noch sehr stark an dem eigentlichen HTML, und zum Auslesen entsprechender Parameter ist einiges an Aufwand notwendig.

Auch das Setzen eines Cookies erinnert mehr an die Grundsätze des HTTP-Protokolls:

```
001 #!/usr/bin/perl
002
003 #Setzen von Cookies
004 print "Set-Cookie:Nutzer=XYZ;\n";
005 print "Set-Cookie:Passwort=XYZ123;\n";
006 print "Set-Cookie:Expires=Tuesday, 31-Dec-2018 00:00:00 GMT";\n";
007 print "Set-Cookie:Domain=www.test.url;\n";
008 print "Set-Cookie:Path=/;\n";
009 print "Content-type:text/html\r\n\r\n";
010
011 #Hier würde der Content folgen
```

Im Laufe der Jahre wurde von diesen Schichten in anderen Sprachen immer weiter abstrahiert — nicht so in Perl[9]. Perl findet deshalb heutzutage kaum noch Anwendung.

Weitere Technologien

Die am häufigsten genutzten Endungen und die passenden serverseitigen Technologien sind im Folgenden alphabetisch aufgeführt. Darunter sind auch einige Technologien, die wir aufgrund der geringen Verwendung oben nicht weiter behandelt haben:

[9] Mittels Frameworks wurde das erleichtert.

- `asp` — Microsoft Active Server Pages
- `aspx` — Microsoft ASP.NET
- `cfm` oder `cfml` — Cold Fusion
- `cgi` — Common Gateway Interface
- `dll` — meist C oder C++
- `nsf` oder `ntf` — Lotus Notes
- `jsp` — Java Server Pages
- `php` — PHP
- `pl` — Perl
- `py` — Python

Diese Endungen sind jeweils Teil der URL, anhand der man leicht erkennen kann, welche serverseitige Sprache eine Webapplikation nutzt. Wir werden diese Endungen im Rahmen des Kapitels zu Google-Dorking erneut betrachten (siehe Abschnitt »Fehler! Verweisquelle konnte nicht gefunden werden.«).

Was neben einer Dateiendung eine URL sonst noch ausmacht und wie man mittels URL Parameter übermitteln kann, betrachten wir nach der Einführung in eine besondere Sprache: SQL.

SQL

SQL (Structured Query Language) ist eine Sprache, um Zugriff auf Datenbankserver zu erhalten, etwa MySQL, MS-SQL oder Oracle. Die allermeisten Webapplikationen besitzen heutzutage nachgelagerte SQL-Subsysteme, die mittels SQL dynamische Inhalte verwalten und auslesen können. Dazu werden Daten in Tabellen abgespeichert und sind mittels Spalte abrufbar. Jede dieser Spalten enthält entsprechende Daten, die mit sogenannten SQL-Abfragen (Queries) aufgerufen werden können.

Eine Abfrage ist meist folgendermaßen aufgebaut:

```
001 SELECT "Spalten_Name"
002 FROM "Tabellen_Name"
003 WHERE "Bedingung";
```

Eine simple SQL-Datenbank von Benutzern einer Webapplikation (`users`) könnte beispielsweise folgendermaßen aussehen:

id	name	passwort	email
1	Admin	Super8sicher#	admin@seite.de
2	Rudolf	Sbratwurst32	rudolf@email.de
3	Marie	758Dlomted	marie@email.de
4	Markus	qA%bsolutsicher	markus@email.de
5	Hans	lmp4tw	hans@email.de
6	Test	testsecured	eine@email.de
…	…	…	…
95	Victoria	princessxD	vic@toria.de

Eine typische SQL-Abfrage zu dieser Datenbank sieht so aus:

```
SELECT email FROM users WHERE name = ‘Hans’;
```

In diesem Fall wird »hans@email.de« zurückgegeben.

Die wichtigsten Befehle für das Bilden von Abfragen in SQL sind folgende:

- `SELECT` — Auswahl der Attribute
- `FROM` — Tabellenname
- `WHERE` — Bedingung für die Auswahl
- `GROUP BY` — Gruppieren der Ergebnisse
- `ORDER BY` — Reihenfolge der Ergebnisse

Neben dem Auswählen und Ordnen von Daten aus Tabellen gibt es auch Befehle zum Löschen und Bearbeiten von Daten:

- `UPDATE` — zum Verändern von Daten

```
001 //Beispiel mit UPDATE für die oben angeführte Datenbank:
002 UPDATE users SET email=‘test@test.de‘ WHERE name=‘Test‘;
003
004 //Nach Ausführung dieses Befehls wäre der Nutzer "Test” (id = 6) wie
    folgt:
```

id	name	passwort	email
6	Test	testsecured	test@test.de

- `DELETE` — zum Löschen von Daten

```
001 //Beispiel mit DELETE für die oben angeführte Datenbank:
002 DELETE FROM users WHERE name='Markus';
003
004 //Nach Ausführung dieses Befehls würde die Datenbank wie folgt
    aussehen:
```

id	name	passwort	email
1	Admin	Super8sicher#	admin@seite.de
2	Rudolf	Sbratwurst32	rudolf@email.de
3	Marie	758Dlomted	marie@email.de
5	Hans	lmp4tw	hans@email.de
6	Test	testsecured	test@test.de
...			
95	Victoria	princessxD	vic@toria.de

Gelöschte IDs werden nicht erneut verwendet, sondern jeder neue Eintrag in einer Tabelle erzeugt eine neue Zeile. Die zwei Beispiele sollten zunächst genügen, um die Funktionalität von SQL und Datenbanken zu erläutern. Insbesondere im Kapitel über SQL-Injections werden wir weitere Befehle von SQL kennenlernen. (siehe Kapitel 6.1, »Fehler! Verweisquelle konnte nicht gefunden werden.«).

Zusammenfassung

Nach der Betrachtung der serverseitigen Sprachen des Webs fällt uns auf, dass man bei einigen Technologien genauer differenzieren muss.

Offensichtlich gibt es nicht nur reine Skriptsprachen im Web, wie etwa PHP oder Perl, sondern auch sogenannte Webframeworks wie ASP.NET, Java und Ruby on Rails. Letztere bieten den entscheidenden Vorteil, dass bereits eine Vielzahl von Klassen vorgesehen ist, um ein simples Programmieren und Erstellen von Webapplikationen zu ermöglichen.

Wie wir zuletzt auch gelernt haben, bieten Datenbanken die Möglichkeit zum Verwalten und Ändern von Daten. Diese nachgelagerten Datenbanksysteme werden meist mittels SQL angesprochen, und häufig verwendete Technologien sind MySQL, MS-SQL und Oracle.

Später werden Sie merken, dass trotz dieser Diversität von Technologien ähnliche Angriffsvektoren auf verschiedene Applikationen möglich sind. Das liegt daran, dass all diese Applikationen auf dem uns bereits bekannten HTTP-Protokoll aufbauen und

zudem grundsätzlich ähnliche Technologien wie etwa SQL oder einen Webbrowser benötigen, um zu funktionieren.

URL

Bis jetzt haben wir uns immer etwas darum gedrückt, zu sagen, wie der Nutzer mit seinem Browser auf eine Webseite kommt. Das Geheimnis dahinter sind sogenannte »Uniform Resource Locator« (URL) — für viele von uns sind sie so alltäglich, dass sie fast keine Rolle mehr zu spielen scheinen. Dennoch sind sie einiges an Erklärung wert, da ihr Aufbau in Bezug auf so manche Sicherheitslücke gefährlich ist. URLs bieten viel mehr Optionen, als sich zunächst erahnen lässt. Sie wurden im RFC 3986 [19] standardisiert.

Wenn wir uns den Aufbau einer URL im Folgenden einmal genau ansehen, können wir in der Syntax der URL verschiedene Teile erkennen (die nicht in eckigen Klammern befindlichen Teile sind obligatorisch):

```
protokoll://login@hostname[:port]/[pfad/]datei[?param=wertfragment]
```

Wir erkennen, dass eine URL aus folgenden Teilen besteht:

1. **Protokoll:** Das Protokoll des Webs ist HTTP bzw. HTTPS (was es damit genau auf sich hat, haben wir bereits in Kapitel 3.1 »HTTP« erfahren). Die meisten modernen Webbrowser vervollständigen das Standardprotokoll nach Eingabe einer URL automatisch und schreiben `http://` bzw. `https://`. Nach dem Protokoll folgt immer `://`.
2. **Log-in:** `login` bezeichnet in diesem Fall die Beschreibung eines Benutzers zur Authentifizierung. Wenn Webseiten passwortgeschützt sind, kann man einen Benutzer angeben, mit dem man sich anmelden möchte. Zusätzlich würde eine Passwortanfrage im Browser-Pop-up erscheinen. Diese Funktion wird heute kaum genutzt, der Großteil des Webs ist öffentlich einsehbar oder benutzt andere Möglichkeiten der Authentifizierung.
3. **Hostname:** Der Hostname kann entweder aus einer IP-Adresse (etwa `127.0.0.1`, was dem gerade lokal genutzten Rechner entspricht) oder einer Domäne (engl. Domain) bestehen. Für Menschen sind Buchstaben und Namen deutlich leichter zu merken als Zahlen, deshalb hat sich die Nutzung von Domains etabliert. Mit dem Hostnamen wird eine eindeutige Quelle beschrieben. Beim Aufruf einer URL löst der Browser den Hostnamen auf und schaut nach, welchem Server er eine Anfrage schicken muss.[10]

[10] Die Auflösung erfolgt über das DNS-Protokoll (Domain Name System). Da es kein Protokoll der

4. **Port:** Der Port kann, muss aber nicht, genauer beschrieben werden. Standardmäßig nutzen alle Webbrowser Port 80, der extra für das Protokoll des Webs (HTTP) reserviert wurde.

5. **Pfad:** Nach dem Port bzw. dem Hostnamen kann ein Pfad folgen, der wie im Explorer oder Dateimanager eine Art Unterverzeichnis bezeichnet. Damit lassen sich also Unterverzeichnisse gezielt ansprechen. Viele Webseiten haben mittlerweile sogenannte Sitemaps, über die sich alle Unterseiten finden lassen und in denen die Struktur der Website erkennbar ist. Dennoch ist der Pfad nach wie vor wichtig, um Webseiten eine gewisse Struktur und Tiefe zu geben. Entscheidend ist, dass man den Pfad mit einem `/` einleitet und beendet, um ihn klar von anderen Bestandteilen der URL abzugrenzen.

6. **Datei:** Nach dem Pfad folgt die eigentliche Datei. Ähnlich wie im Explorer spricht man genau ein Dokument an, das man öffnen oder in diesen Fall im Browser angezeigt haben möchte. Meist hat diese Datei eine Endung, etwa `.html`. Damit ist eindeutig, welchem Typ sie entspricht, und der Browser kann dieses HTML-Dokument und interpretieren, um es dann auszugeben.

7. **Fragezeichen:** Mit einem Fragezeichen können an jede URL ein oder mehrere Parameter angehängt werden. Verbunden werden mehrere Parameter mit einem `&`-Zeichen. Den Parametern weist man mit `=` einen Wert zu, jeder Parameter kann nur einen Wert haben.

8. **Fragmente:** Sie dienen dazu, beispielsweise einen zusätzlichen Wert zu übertragen. Es ist gängig, darüber etwa Absätze als »Anker« zu setzen. Wenn man auf eine URL mit einem entsprechenden Fragment geht, das immer mit `#` eingeleitet werden muss, landet man nach dem Laden der Seite direkt im entsprechenden Absatz.

Nachfolgend ist eine typische URL abgebildet:

```
http://eine.webseite/verzeichnis/test.php?hallo=true&a=1#Absatz2
```

Die Adresse (der Hostname) entspricht in diesem Fall keinem bekannten Standard[11] bzw. ist nicht auflösbar für den Browser. Man könnte anstelle dieses ausgedachten Hostnamens natürlich auch einen funktionstüchtigen nehmen, dann würde der Browser den Hostnamen auflösen und den Server um die Übersendung der beschriebenen Datei bitten. In diesem Fall würde der Zugriff mit dem gängigen angegebenen Protokoll `http` nach der in der URL bezeichneten PHP-Datei `test.php` im Verzeichnis `verzeichnis` auf dem entsprechenden Server gefragt und die Parameter würden

Anwendungsebene ist, werden wir es hier nicht tiefergehend behandeln.

[11] Es gibt die Möglichkeit, Domains zu registrieren, allerdings gibt es nur einen begrenzten Spielraum. Übliche Domains haben Endungen (Top-Level-Domains) wie *.de/.org/.com*.

übermittelt. Nach Aufruf der Webseite würde man sich direkt im zweiten Absatz befinden (siehe Fragment), und der Server hätte einen Parameter `hallo` mit dem Wert `true` und einen Parameter `a` mit dem Wert `1` erhalten.

Wie an dem Beispiel deutlich werden sollte, werden nicht erwähnte Teile nicht übermittelt (beispielsweise das Log-in). Es wäre auch möglich, die Parameter einfach wegzulassen, dann erhält der Server allerdings nur eine Art Basisanfrage und kann die Werte logischerweise nicht berücksichtigen, da er sie nicht kennt.

Domains sind nach strengem Standard definiert. So waren lange Zeit keine Umlaute zulässig, mittlerweile sind sie, wie auch das scharfe s (ß), bei den meisten Vergabestellen erlaubt. Leerzeichen sind unzulässig, und die Trennung von Wörtern kann nur durch Bindestrich erfolgen. Während es bei Domains eine Art Regulierung durch Vergabe- und Registrierungsstellen gibt, fehlt diese bei URLs gänzlich. Mittlerweile ist ein ziemliches Durcheinander entstanden, welcher Webbrowser zu was in der Lage ist.

Browser können URLs verschleiern

Es gibt zum Beispiel einige Browser, bei denen man die URL gezielt verschleiern kann, etwa indem man das Oktalsystem[1] nutzt.

Nachfolgend sind einige findige Tricks, um das Ziel einer URL zu verschleiern, aufgeführt:

- Eine Adresse wie *http://0x7f.1/* wird von manchen Browsern beispielsweise korrekterweise als *http://127.0.0.1* gelesen. Dabei wird der erste Teil der IP-Adresse, die 127, durch eine Hexadezimalzahl und der Rest als eine einzige Dezimalzahl wiedergegeben.
- Immer noch sehr beliebt, um als Angreifer Nutzer auf eine falsche Seite zu locken, ist, die Anmeldedaten so zu präparieren, dass sie auf den ersten Blick wie eine gültige URL wirken. Wohin beispielsweise, denken Sie, führt die URL *http://paypal.com&test=test=123@2915201127*? Ein Großteil der Nutzer würde sicherlich davon ausgehen, dass der Link zu *paypal.com* führt — das ist jedoch falsch, Sie landen auf *google.com*.

  Warum? Wir haben gerade eben gelernt, dass mittels `@`-Zeichen der entsprechende Rechner angesprochen wird, in diesem Fall ist die `2915201128` eine Integer-Darstellung einer Google-IP-Adresse.

Diese »Lücke« wurde in manchen Browsern (etwa dem Mozilla Firefox) mittlerweile mit einem Hinweis versehen — in anderen ist aber ein Aufruf ohne Bestätigung denkbar oder eine Auflösung der URL nicht möglich.

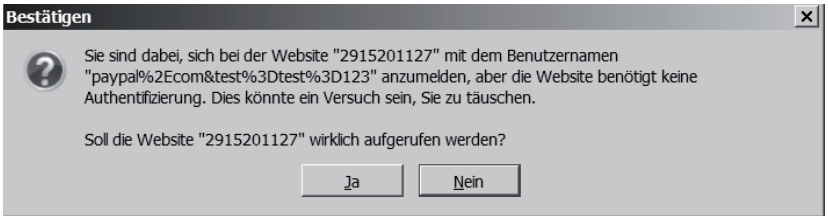

Bild 9: Hinweis zum URL-Aufruf im Mozilla Firefox.

- Normale Linkkürzungsdienste wie *bit.ly* bieten Angreifern immer wieder die Möglichkeit, den eigentlichen Inhalt hinter einer URL zu verstecken, allerdings sinkt bei solchen URLs die Wahrscheinlichkeit, dass der Nutzer den Link anklickt, da er möglicherweise Lunte riecht oder einmal gehört hat, dass man nicht ohne Weiteres auf unbekannte Links klicken sollte.
- »Sicherer« ist es für Angreifer, Domains zu nutzen, die ähnlich wie die normalen Domains eines Diensts aussehen. Statt eines o kann beispielsweise eine 0 (Null) in einem Link genutzt werden oder einfach eine komplett neue Top-Level-Domain. Dieses Vorgehen nennt sich Domainsquatting und wird häufig auch bei Links in Phishing-E-Mails angewandt.

Wir wissen nun einiges mehr über URLs. Neben der Verwendung bei Verlinkungen im Quellcode (siehe Abschnitt »Links«) spielen URLs vor allem bei dem Abruf von Webseiten eine wichtige Rolle. Im Buch werden wir auf einige Tricks bezüglich URLs zu sprechen kommen (siehe dazu Kapitel 12.3, »Fehler! Verweisquelle konnte nicht gefunden werden.«).

Der häufigste Einsatz von URLs ist wohl das Eintippen in der Adresszeile eines Webbrowsers, und genau das wollen wir nun näher betrachten.

Webbrowser

Der bloße HTML-Quelltext einer Webseite bringt einem normalen Nutzer rein gar nichts. In ihm befinden sich ausschließlich Tags, die mit spitzen Klammern Ausdrücke definieren, angenehm lesen lässt sich der Quellcode auch nicht. Erst ein Programm, der Webbrowser, interpretiert HTML-Quellcode, bindet Medien (etwa Bilder) ein und liefert dem Benutzer eine grafische Aufbereitung der übermittelten Informationen. Für die angenehme Nutzung des Webs war und ist also ein Webbrowser notwendig.

Der erste Webbrowser wurde »WorldWideWeb« genannt und 1990 von Tim Berners-Lee entwickelt [20], später war dieser Browser nicht mehr von Bedeutung.

Es ist kaum verwunderlich, dass sich durch die neu entstandene Marktlücke der sogenannte »Browserkrieg« entwickelt hat. Dabei handelt es sich um einen Verdrängungswettbewerb der Browserhersteller Microsoft und Netscape, der Mitte der 1990er-Jahren seinen Höhepunkt erreichte. Microsoft hatte die Auswirkungen des Webs schlicht unterschätzt und wollte diesen Rückstand aufholen. Der Netscape Navigator erreichte zeitweise einen Marktanteil von fast 80 %. Um Marktanteile zu gewinnen, investierte Microsoft stark in die Entwicklung und das Marketing eines eigenen Browsers (des Internet Explorers). Es gelang Microsoft schließlich, Marktführer zu werden und das Verhältnis zu drehen.

Mit dem Marktanteil eines Webbrowsers geht ein großer Einfluss einher. Schließlich wäre man durch eine Monopolstellung in der Lage, standardmäßig Startseiten oder Suchmaschinen vorzugeben oder sogar gewisse Standards im Web zu beeinflussen.

Heute gibt es viele verschiedene Browser, alle jedoch folgen der zugrunde liegenden Idee eines Übersetzers für den HTML-Quelltext. Dieser wird HTML-Renderer genannt. Trotz vieler Standards gehen manche HTML-Renderer mit bestimmten Inhalten anders um als andere und interpretieren HTML-Elemente geringfügig unterschiedlich, als Entwickler kann einen das bekanntermaßen in den Wahnsinn treiben.

Im Internet Explorer gibt es seit Version 5 sogenannte »Conditional Comments«. Da häufiger Kompatibilitätsprobleme mit CSS-Code im Internet Explorer auftraten, entwickelte Microsoft dieses Konzept, um Abhilfe zu schaffen. Im Quellcode sieht das Ganze etwa so aus:

```
001 <style type="text/css">@import url(normale_browser.css) all;</style>
002
003 <!–[if IE]>
004    <style type="text/css">@import url(ie.css);</style>
005 <![endif]–->
006
007 <!–[if IE 6]>
008    <style type="text/css">@import url(ie6.css);</style>
009 <![endif]–->
```

Der Auszug aus der HTML-Datei zeigt, dass zunächst eine normale CSS-Datei geladen wird, anschließend folgen Kommentare. Diese Kommentare werden vom Internet Explorer allerdings nicht, wie eigentlich üblich, überlesen, sondern interpretiert, wenn die in den Klammern enthaltenen Bedingungen zutreffen. So ist es möglich, ausschließlich CSS-Anweisungen für den Internet Explorer zu platzieren. Andere Browser überlesen die »Conditional Comments« schlichtweg.

Die bekanntesten HTML-Renderer sind:

- **Gecko** (Mozilla Firefox)
- **WebKit/Blink** (Safari, Chrome)
- **Trident** (Internet Explorer)

Worin sie sich genau unterscheiden, soll an dieser Stelle nicht das Thema sein, stattdessen wollen wir nur kurz die grundlegende Funktionsweise betrachten.[12]

Alle HTML-Renderer dienen dazu, ein sogenanntes Parsing vorzunehmen, also das erhaltene HTML einzulesen und zu strukturieren (dabei werden teilweise auch eventuelle Syntaxfehler korrigiert). Nach dem Einlesen wird das sogenannte DOM (Document Object Model) erstellt, um Zugriffe mittels JavaScript zu ermöglichen und die Webseite anschließend (unter Berücksichtigung von CSS und anderer Technologien) auszugeben.

Zu HTML und CSS sind über die Jahre sehr viele weitere Plug-ins oder Skriptsprachen, die ebenfalls interpretiert werden können, hinzugekommen (siehe Abschnitt »Weitere clientseitige Technologien«). Diese werden nicht serverseitig, sondern beim Aufruf der Webapplikation im Browser lokal ausgeführt. So gibt es zum Beispiel sogenannte JavaScript-Engines, die, wie der Name schon sagt, dafür zuständig sind, dass das JavaScript über Schnittstellen Zugriffe auf das DOM erhält, also die Inhalte einer Webseite verändern kann.

Heute nutzen wir Browser täglich, und sie stehen auf nahezu jedem neu installierten Betriebssystem als »Tor zum Web« zur Verfügung. In den Anfangszeiten des Webs waren sie in keiner Standardinstallation von Betriebssystemen enthalten und mussten nachinstalliert werden. Auch wenn immer mehr alternative Technologien Zugriffe auf das Web ermöglichen, etwa Apps oder »smarte« Geräte, werden Webbrowser wohl noch lange Zeit überdauern.

Die bekanntesten und am weitesten verbreiteten Webbrowser sind:

- Mozilla Firefox
- Chrome
- Safari
- Internet Explorer
- Opera
- Microsoft Edge

Die Beispiele im Buch werden meist mit dem Mozilla Firefox gezeigt. Generell sind sie aber in jedem Browser umsetzbar, und wir werden auch auf die Spezifikation in anderen Browsern zu sprechen kommen.

[12] Da sich dieses Buch nicht in erster Linie mit Webbrowsern, sondern mit Webapplikationen beschäftigt, ist die Verschiedenheit der Renderer ein hier nicht weiter zu behandelndes Thema.

Document Object Model (DOM)

Wir haben gelernt, dass JavaScript auf Cookies zugreifen kann. Dieser Zugriff wird über das DOM realisiert. Das DOM ist eine Schnittstelle im Browser, um den abstrakten Zugriff auf HTML-Dokumente zu ermöglichen, der Zugriff erfolgt meist über eine entsprechende ID.

Mittels DOM wird im Speicher eine strukturierte Darstellung des aktuell geladenen HTML-Dokuments (oder XML) angelegt (sogenanntes Parsing), das zuvor durch den HTML-Renderer ermittelt wurde. Zur Darstellung des aktuellen Dokuments kommt immer das DOM zum Einsatz und nicht das ursprüngliche HTML-Dokument. Man könnte also das ursprüngliche Dokument als Quelle verstehen (in der sich eventuell noch Fehler befinden) und die Darstellung mittels DOM als fertige (korrigierte) Form.

Das DOM besitzt eine Vielzahl von Eventhandlern, die wichtigsten sind nachfolgend aufgeführt:

- `document.head` — Gibt den Header `<head>` zurück.
- `document.body` — Gibt das Body-Element `<body>` zurück.
- `document.cookie` — Gibt alle Cookies des Dokuments zurück.
- `doument.domain` — Gibt die Domain des Servers zurück, der das Dokument übermittelt hat.
- `document.referer` — Gibt den Referrer des aktuellen Dokuments zurück.
- `document.URL` — Gibt den gesamten Pfad des HTML-Dokuments zurück.
- `document.title` — Gibt den Seitentitel des Dokuments zurück.
- `document.open()` — Öffnet ein neues HTML-Dokument, um Inhalte von `document.write()` aufzunehmen.
- `document.close()` — Schließt das HTML-Dokument nach `document.open()` wieder.
- `document.write()` — Schreibt HTML bzw. JavaScript in ein Dokument.

Besonders genannt seien an dieser Stelle `getElementById()` und `getElementByName()`, mit denen man betreffende Elemente aus dem DOM-Baum ermitteln und mithilfe der ID oder des Namens auch ansprechen kann. Diese Funktionen nutzen beispielsweise JavaScript-Engines, die für die Ausführung von JavaScript-Code zuständig sind.

Browsersicherheit

Unter dem Sicherheitsaspekt ist die Anatomie von Browsern sehr interessant. Mittlerweile gibt es eine ganze Sammlung von Sicherheitsmechanismen, die sich jedoch von Browser zu Browser unterscheiden. Wir werden uns mit unterschiedlichen Methoden, etwa dem Setzen von besonderen HTTP-Headern, noch beschäftigen.

Wie bekannt ist, besteht beispielsweise mittels Tabs die Möglichkeit, zeitgleich verschiedene Applikationen zu öffnen. Der Zugriff auf andere Sitzungen oder DOM-Inhalte anderer Webseiten darf allerdings nicht möglich sein, da sonst jegliche Sicherheitsprinzipien durchbrochen wären. Für jeden Browser gilt also: Inhalte müssen voneinander isoliert werden. Dazu gibt es einen zentralen, für alle modernen Browser gleichlautenden Ansatz: die sogenannte Same-Origin-Policy, die wir nun betrachten.

Same-Origin-Policy (SOP)

Wie der Name schon sagt, geht es bei dem Sicherheitskonzept darum, mithilfe des Ursprungs (Origin) Prinzipien festzulegen. Die SOP wurde 1996 eingeführt und bestimmt konkret, welche Rechte Inhalten, zum Beispiel clientseitigen Skripten wie JavaScript, eingeräumt werden.

Laut SOP dürfen Skripte nur jene Inhalte lesen und verändern, die ihrem eigenen Ursprung entstammen. Der Zugriff auf die Inhalte anderen Ursprungs ist nicht ohne Weiteres erlaubt.

Der Ursprung wird als Kombination aus folgenden Daten definiert:

- Protokoll (HTTP oder HTTPS)
- Domain
- Port

Nur wenn alle drei Merkmale gleich sind, ist ein Zugriff laut SOP zulässig. Die folgende Tabelle verdeutlicht das Konzept hinter der SOP.

Es wird davon ausgegangen, dass ein Skript, das unter *http://www.test.url/try/sop.html* eingebettet ist, versucht, Zugriff auf folgende Ressourcen zu erhalten:

Aufgerufene URL	Zugriff	Grund
http://www.test.url/try/index.html	zulässig	Protokoll und Host gleich
http://www.test.url/testing/test.html	zulässig	Protokoll und Host gleich

Aufgerufene URL	Zugriff	Grund
http://sub.test.url/test/test.html	unzulässig	Host ist ungleich
http://test.url/test/try.html	unzulässig	Host ist ungleich
http://www.test.url:21/tested/try.html	unzulässig	Port (21) ist ungleich
https:// www.test.url/try/index.html	unzulässig	Protokoll ist ungleich

Eine Ausnahme ist die explizite Nennung des eigentlich korrekten Ports in der URL. Beispielsweise wird ein Zugriff auf *http://www.test.url:80/try/sop.html* von manchen Webbrowsern als zulässig betrachtet, andere verwerfen ihn allerdings als unzulässig.

Subdomains bilden eine Ausnahme von der SOP, grundsätzlich ist der gegenseitige Zugriff, wie wir der Tabelle oben entnehmen können, unzulässig. Allerdings ist es möglich, von einer Subdomain auf die Elemente der Domains zu zugreifen. *test.url.de* kann also *url.de* beeinflussen, beispielsweise indem man folgendes Skript anwendet:

```
document.domain = "url.de";
```

Nun wäre der Zugriff auf die Elemente von *url.de* möglich. Das Setzen der Domain kann jedoch nicht beliebig erfolgen, beispielsweise wäre das Setzen von

```
document.domain = "andereurl.de";
```

unter *test.url.de* nicht möglich.

Auch mit CORS (Cross-Origin Sharing Standard) ist ein Zugriff zwischen verschiedenen Domains möglich, dazu wird das HTTP-Headerfeld `Access-Control-Allow-Origin` bei HTTP-Antworten hinzugefügt. SOP und CORS finden unter anderem bei der Verarbeitung von Cookies häufig Verwendung. Die SOP stellt wohl das bedeutendste Sicherheitselement für moderne Browser und somit auch Webanwendungen dar. Wir werden später betrachten, was passiert, wenn die SOP durchbrochen wird oder das `Access-Control-Allow-Origin`-HTTP-Headerfeld sogar unfreiwillig folgenden Wert (Wildcard-Eintrag) annimmt:

```
Access-Control-Allow-Origin: *
```

Zuvor betrachten wir allerdings noch einmal das Thema der Codierung.

Codierung

Sowohl HTML als auch HTTP basieren ursprünglich auf reinem Klartext. Mit der Zeit haben sich allerdings verschiedene Arten der Codierung entwickelt.

Codierungen dienen als Übertragungsverfahren, durch sie können Inhalte manchmal kompakter oder sogar komprimiert übermittelt oder »verschleiert« werden. Meist sind Codierungsarten allerdings einfach nur eine andere Darstellungsart.

Da wir uns mit der Sicherheit von Webanwendungen detaillierter beschäftigen wollen, ist es sinnvoll, dass wir uns mit den Möglichkeiten der Codierung von Inhalten näher auseinandersetzen, denn manchmal kann schon die Änderung der Codierung dazu führen, dass eine Webapplikation ungewünschte Aktionen ausführt. Andersherum kann »Verschleierung« (nach dem Prinzip »Security through Obscurity«[13]) allerdings auch helfen, Daten zu verschleiern.

Wird von Klartext in Codierung umgewandelt, ist die Rede von codieren (Codierung), wird dagegen ein codierter Inhalt wieder in Klartext umgewandelt, spricht man von decodieren (Decodierung).

ASCII-Zeichensatz

Eine wichtige Basis für die meisten Codierungsarten ist der sogenannte American Standard Code for Information Interchange (ASCII). Dabei handelt es sich um die Festlegung eines einheitlichen Zeichensatzes aus dem Jahr 1963, der zunächst nur für die Informationsübermittlung in den USA gedacht war. Mittlerweile gilt er, neben anderen Zeichensätzen, als internationaler Standard.

Später wurde ASCII von Unicode, einem weitaus größeren Zeichensatz, weitgehend abgelöst. Der Unicode enthält den kompletten ASCII, mit ihm ist bis heute die Darstellung von nahezu allen Zeichen der Welt möglich.

Die wichtigsten Codierungsverfahren wollen wir nachfolgend einmal betrachten.[14]

HTML-Codierung

Stellen wir uns zunächst einmal die Frage: Wie kann man eine Webseite erstellen, auf der man HTML lernen kann, also die Tags entsprechend erläutert bekommt, ohne dass diese vom Browser interpretiert werden?

[13] Das Konzept hinter »Security through Obscurity« (Sicherheit durch Unklarheit) ist umstritten und sollte allenfalls als zusätzlicher Schutz dienen.

[14] Ein grundlegendes Verständnis der Codierungsmethoden ist für die später gezeigten Angriffe durchaus nützlich. Siehe Literaturempfehlung: »Algebraische Grundlagen der Informatik«.

Hier bietet die HTML-Codierung einen Ansatz, bei dem die vorbelegten und in HTML angewendeten Zeichen durch sogenannte namentliche Entitäten ausgetauscht werden. In der Darstellung gleichen sie den uns bekannten Zeichen, allerdings werden sie nicht vom Browser interpretiert.

Bekannte Beispiele für diese Art der Codierung sind zum Beispiel Anführungszeichen. Neben dieser namentlichen Art der Darstellung ist es auch möglich, den gleichen Effekt zu erzielen, indem man den ASCII-Code in dezimaler Form anspricht.

Zeichen	Entsprechende namentliche (X)HTML-Entität	Entsprechende HTML-Entität (mittels ASCII)
"	&quot;	&#34;
'	&apos;	&#39;
<	&lt;	&#60;
>	&gt;	&#62;
&	&amp;	&#38;

Eine besondere Ausnahme in dieser Tabelle ist, dass ' (bzw. `&apos;`) ausschließlich in XHTML, nicht aber in HTML, verwendet werden kann.

Ein weiterer Trick zur Darstellung, den wir in ähnlicher Form bereits bei URLs (siehe Kapitel 3.3, »URL«) angewandt haben, ist die Nutzung von hexadezimalen Zahlen. Dabei kann die entsprechende ASCII-Nummer (die der Position im ASCII-Code entspricht) in das Hexadezimalsystem umgewandelt werden. Ein simples Anfügen eines `x` vor der Hexadezimalzahl reicht dafür aus (mehr dazu gleich in Kapitel 3.5.6, »Hexadezimale Codierung«).

Zeichen	Entsprechende HTML-Entität (mittels ASCII)	Hexadezimale Ansprache des ASCII-Codes
"	&#34;	&#x22
&	&#38;	&#x26

Das Beispiel der HTML-Codierung zeigt uns, dass massenhaft Möglichkeiten bestehen, eine Darstellung zu erreichen. Gerade die sichere Darstellung und Verarbeitung von bereits belegten Zeichen spielen bei der Abwehr von gängigen Angriffsvektoren eine wichtige Rolle, wie wir später sehen werden. Durch die Vielzahl von Varianten gelingt es Angreifern allerdings immer wieder, Wege zu finden, um Filter oder Schutzmaßnahmen zu umgehen — dazu später mehr.

URL-Codierung

URL-Codierung ist, wie der Name schon sagt, eine Codierungsart, die bei der Codierung innerhalb der URL verwendet wird. Wenn wir uns an den Aufbau von URLs erinnern (siehe Kapitel 3.3, »URL«), wissen wir, dass Zeichen wie `@ ? & /` besondere Bedeutungen besitzen, wenn sie in einer URL enthalten sind.

Damit diese Zeichen innerhalb der URL (etwa bei Übergabe von GET-Parametern) nicht fälschlicherweise interpretiert werden, gibt es URL-Codierung. Teilweise findet sich in der Fachliteratur auch der Name Prozentcodierung, was darauf zurückzuführen ist, dass entsprechende in der URL vorbelegte Zeichen ein %-Zeichen am Anfang haben.

Die Codierung sieht folgendermaßen aus:

Am Anfang steht ein Prozentzeichen (`%`), anschließend folgt die Stelle des Zeichens im ASCII-Code in hexadezimaler Darstellung.

Ein = steht im ASCII-Code beispielsweise an der Stelle 61. Diese 61, als Hexadezimalzahl dargestellt, entspricht folgendem Wert: `3D`.

Ein = entspricht also in URL-Codierung: `%3D`.

Folgende Tabelle zeigt einige wichtige Zeichen in URL-Codierung:

Zeichen	URL-Codierung
Zeilenumbruch (New Line)	%0a
(Nullbyte)	%00
(Leerzeichen)	%20
"	%22
%	%25
&	%26
'	%27
=	%3D
?	%3F
@	%40

Zum Angreifen von Webapplikationen ist das Wissen über diese Codierungsmechanismen fast unerlässlich, denn nur wenn ein Angreifer weiß, wie Dinge verarbeitet werden, besteht auch die Möglichkeit, dass er die Verarbeitung in seinem Sinne nutzen kann.

Unicode-Codierung

Unicode ist unter den Zeichensätzen jener, der sämtliche Schriftsätze aller Schriften der Welt darstellen kann — insoweit ist er sehr mächtig. Die ersten 128 Zeichen des ASCII-Codes (0–127) und des Unicodes gleichen sich — danach verfügt der Unicode über Zeichen, die im ACSII nicht vorhanden sind.

Unicode-Codierung funktioniert im Großen und Ganzen wie die zuvor dargestellte URL-Codierung, umfasst aber deutlich mehr Zeichen.

Wenn Unicode genutzt wird, gibt es zahlreiche Möglichkeiten der Codierung, die wir im Folgenden einmal alle am Beispiel des Euro-Zeichens betrachten wollen:

Zeichen	€
HTML-Entität (dezimal)	&#8364;
HTML-Entität (hexadezimal)	&#x20ac;
HTML-Entität (namentlich)	&euro;
Windows	Alt + 20AC
UTF-8 (hexadezimal)	0xE2 0x82 0xAC (e282ac)
UTF-8 (binär)	11100010:10000010:10101100
UTF-16 (hexadezimal)	0x20AC (20ac)
UTF-16 (dezimal)	8.364
UTF-32 (hexadezimal)	0x000020AC (20ac)
UTF-32 (dezimal)	8.364
C/C++/Java-Quelltext	"\u20AC"
Python Source Code	u"\u20AC"

UTF-8

Der gängige Standard zum Ansprechen des Unicodes im Web ist UTF-8. Die Ansprache des Zeichensatzes ist auf verschiedene Weise möglich (siehe oben) und von der jeweiligen Programmiersprache bzw. Systemumgebung abhängig.

Laut »W3 Techs Web Technology Surveys« sind mittlerweile über 80 % der Webseiten UTF-8-codiert.

Es gibt zahlreiche weitere Standards, die allerdings rückläufig sind. Man kann davon sprechen, dass der Unicode bzw. »verwandte« Codierungsmechanismen (wie UTF-8) aufgrund der einfachen Handhabung und doch umfassenden Möglichkeit zur Darstellung vieler Zeichen im Web als vorherrschende Technologie genutzt wird.

Der UTF-8-Zeichensatz kommt zum Beispiel auch bei der Darstellung von Umlauten im Deutschen zum Einsatz.

Webseitenbetreiber können über eine sogenannte Zeichencodierung (Charset) festlegen, wie mit den Zeichen auf der Webseite umgegangen wird. Dazu reicht die Einbindung des Zeichensatzes im Metatag (siehe Abschnitt »HTML«).

base64-Codierung

base64-Codierung ermöglicht die Codierung von Dateien in 64 Zeichen des ASCII-Codes.

Dieser Auszug aus dem ASCII-Zeichensatz sieht wie folgt aus:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
```

base64 findet bis heute überwiegend im Bereich des Mailversands über das Simple Mail Transfer Protocol (SMTP) Verwendung. Praktisch jede E-Mail und auch jeder Anhang (etwa ZIP-Dateien, Bilder etc.) wird vor dem Versand in base64 umgewandelt.

Bei der Umwandlung von normalem Klartext in base64 vergrößert sich die Menge um etwa 33 %. Bei der Umwandlung in base64 kann es zudem sein, dass = (Gleichheitszeichen) am Ende als »Füllbytes« eingefügt werden.

Eine base64-Codierung des Texts

```
Nach den Codierungsarten geht es mit Angriffen weiter
```

lautet in base64 folgendermaßen:

```
TmFjaCBkZW4gS29kaWVydW5nc2FydGVuIGdlaHQgZXMgbWl0IEFuZ3JpZmZlbiB3ZWl0ZXI=
```

base64 wird in vielen Webapplikationen eingesetzt, um Cookies oder andere Parameter zu verschleiern. So etwas sieht folgendermaßen aus:

Bild 10: Verschleierung von GET-Parametern.

In diesem Fall wird ein Teil einer URL mittels base64 verschleiert, mit einer einfachen Webapplikation (wie *www.base64decode.org*) oder entsprechenden Tools lässt sich aus

```
Text in base64:
bD1kZSZjPTEwMA==
```

allerdings sehr schnell wieder

```
Text:
l=de&c=100
```

ermitteln.

Der Inhalt selbst ist mit einer bloßen Umkehrung der Codierung zu ermitteln und eignet sich deshalb nicht, um Daten vor dem Client (und somit auch vor klugen Angreifern) zu »verstecken«.

Hexadezimale Codierung

Analog zu base64 kann man Daten auch hexadezimal codieren. Dabei wird oftmals zunächst der Text in die ASCII-Werte und dann in das Hexadezimalsystem umgewandelt.

Zeichenkette	Entsprechender ASCII-Code	Hexadezimale Codierung
T E S T	84 69 83 84	54 45 53 54

Zusammenfassung

Alle diese Varianten der Codierung werden an der einen oder anderen Stelle in diesem Buch wieder auftauchen. Wir werden merken, dass uns Entwicklern die Möglichkeit, Dinge zu codieren bzw. anders darzustellen, an so mancher Stelle helfen und sogar unsere Applikation schützen wird — allerdings kann die Möglichkeit auch für Angreifer sehr nützlich sein, um von uns (unzureichend) gesetzte Filter zu umgehen oder durch Decodierung zu versuchen, der Funktionsweise der Applikation auf die Schliche zu kommen.

Wichtig sei an dieser Stelle noch, dass Codierung selbst keinerlei Schutz bietet, wenn es darum geht, Geheimnisse wie Passwörter sicher zu übertragen oder zu hashen! Dazu sollte immer eine verschlüsselte Verbindung mittels HTTPS (siehe Kapitel 4.6.1, »Fehler! Verweisquelle konnte nicht gefunden werden.«) oder entsprechende Kryptomethoden (siehe Kapitel 7.3, »Fehler! Verweisquelle konnte nicht gefunden werden.«) genutzt werden.

Zunächst einmal: Gratulation!

Sie haben soeben die Grundlagen des Webs kennengelernt, und wir sind nun bereit, einen Schritt weiterzugehen und die ersten tatsächlichen Angriffsvektoren zu betrachten.

(Footnotes)

1 Das Oktalsystem ist ein Stellenwertsystem mit der Basis 8 statt wie gewöhnlich 10.