Edition <kes>

Matthias Rohr

Sicherheit von Webanwendungen in der Praxis

Wie sich Unternehmen schützen können – Hintergründe, Maßnahmen, Prüfverfahren und Prozesse

<kes>

Springer Vieweg

Edition <kes>

Herausgegeben von
P. Hohl, Ingelheim, Deutschland

Mit der allgegenwärtigen Computertechnik ist auch die Bedeutung der Sicherheit von Informationen und IT-Systemen immens gestiegen. Angesichts der komplexen Materie und des schnellen Fortschritts der Informationstechnik benötigen IT-Profis dazu fundiertes und gut aufbereitetes Wissen.

Die Buchreihe Edition <kes> liefert das notwendige Know-how, fördert das Risikobewusstsein und hilft bei der Entwicklung und Umsetzung von Lösungen zur Sicherheit von IT-Systemen und ihrer Umgebung.

Herausgeber der Reihe ist Peter Hohl. Er ist darüber hinaus Herausgeber der <kes> – Die Zeitschrift für Informations-Sicherheit (s. a. www.kes.info), die seit 1985 im SecuMedia Verlag erscheint. Die <kes> behandelt alle sicherheitsrelevanten Themen von Audits über Sicherheits-Policies bis hin zu Verschlüsselung und Zugangskontrolle. Außerdem liefert sie Informationen über neue Sicherheits-Hard- und -Software sowie die einschlägige Gesetzgebung zu Multimedia und Datenschutz.

Weitere Bände in dieser Reihe
http://www.springer.com/series/12374

Matthias Rohr

Sicherheit von Webanwendungen in der Praxis

Wie sich Unternehmen schützen können – Hintergründe, Maßnahmen, Prüfverfahren und Prozesse

Springer Vieweg

Matthias Rohr
Hamburg
Deutschland

Edition <kes>
ISBN 978-3-658-03850-2 ISBN 978-3-658-03851-9 (eBook)
DOI 10.1007/978-3-658-03851-9

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Springer Vieweg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Fachmedien Wiesbaden ist Teil der Fachverlagsgruppe Springer Science+Business Media (www.springer.com)

Geleitwort

Webanwendungen sind omnipräsent. Jeder von uns, der sich eines Rechners und des Internets bedient, benutzt sie ständig: sei es zum Buchen einer Reise, zum Schreiben von E-Mails oder beim Spielen mit Apps auf dem Smartphone. Manchmal sieht man mehr, manchmal weniger von ihnen. Webanwendungen werden aber auch in der Wirtschaft eingesetzt – sei es im Bankenumfeld, beim Warennachschub oder gar bei kritischen Infrastrukturen zur Abbildung von unternehmensübergreifenden Geschäftsprozessen. Ohne Webanwendungen geht nichts mehr – Webanwendungen gehören die Zukunft.

Woran liegt das? Sind Webanwendungen inzwischen die einzige Technologie, die Entwickler noch kennen? Kann man damit so leicht programmieren? Viele von uns haben schon selbst Webseiten geschrieben, da kann eine Webanwendung ja nicht viel schwerer sein. Das stimmt aber nicht. Gute Webanwendungen zu schreiben, ist schwer, viel schwerer, als die klassischen „Client-Server"-Applikationen. Bei Webanwendungen bekommt man nichts abgenommen und muss sich um alle möglichen Aspekte (wie etwa Usability, Benutzermanagement, Input-Validierung) selber kümmern. Natürlich gibt es Frameworks – aber diese muss man kennen, beherrschen und verwalten. Zudem lebt eine Webanwendung von Interaktionen der Benutzer, die durch das http-Protokoll nicht unterstützt werden. Die Nutzeraktivitäten, Sessions genannt, müssen dem http-Protokoll förmlich aufgezwungen werden. Schließlich arbeiten alle Browser-Hersteller ständig an neuen Erweiterungen, die man, um eine optimale Usability erreichen zu können, alle kennen und beherrschen muss.

Der Grund für die häufige Verwendung von Webanwendungen ist – wie so oft – ein rein wirtschaftlicher. Um IT-Anwendungen von verschiedenen Firmen miteinander zu verbinden, müssen diese miteinander kommunizieren. Da sich das Web-Browsing (also das reine Anschauen von Informationen) via HTTP etabliert hatte, wurden zunehmend auch Anwendungen – erst einfache, dann immer kompliziertere – auf dieses Protokoll verlagert. Dies hatte den Vorteil, dass man keine Software für die Clients mehr auf die PCs der Anwender bringen musste, und so Änderungen an der Software der Oberflächen, im Vergleich zu früher, deutlich kostengünstiger realisieren konnte. Ähnliches ist bei den Apps für Smartphones passiert! Apps könnten eigene, spezifische Protokolle verwenden, was manche auch tun, aber die meisten Netze erlauben die Verwendung des HTTP-Ports ebenso für „fremde" Geräte. Daher haben viele Entwickler ihre Apps so gebaut, dass sie

wie Webanwendungen (mit „vorgeladenen“ Webseiten sozusagen) funktionieren. Somit ist der moderne IT-Softwareprogrammierer und Lösungsexperte im Wesentlichen ein Web-Anwendungs-Entwickler.

Diese Vereinheitlichung hat zwar finanzielle Vorteile, bringt aber eine Reihe von Herausforderungen mit sich. Neben den genannten zusätzlichen Aspekten, an die ein Web-Anwendungsentwickler denken muss, ist aufgrund der Offenheit des Protokolls (im Prinzip ist jede Webanwendung vom Internet aus angreifbar) die Sicherheit ein ganz erheblicher Erfolgsfaktor. Doch auch wenn man die „einfachen“ Dinge behoben hat, ist es noch ein langer Weg zu einer „sicheren“ Webanwendung. Die Konsequenzen sind fatal: neben „nur“ fehlender Compliance wird mit mangelnder Sicherheit von Webanwendungen der Internetkriminalität Tür und Tor geöffnet, das Vertrauen von Kunden und Partnern untergraben und schlimmstenfalls das Geschäftsmodell ad absurdum geführt.

Wie kann man dem begegnen? Gute Empfehlungen für Web-Anwendungsentwickler sind im Netz im Überfluss vorhanden – so viel, dass es schon wieder schwer ist, die Spreu vom Weizen zu trennen. Damit kann man zwar die Entwickler erreichen, aber ohne die entsprechende Unterstützung durch das Management sind die erforderlichen Maßnahmen, aufgrund der damit verbundenen Aufwände und Investitionen, nicht umsetzbar. Sicherheit – dies gilt für Webanwendungen genauso wie für alle anderen Sicherheitsthemen oder -bereiche – ist nur dann realisierbar, wenn es auch ein Verständnis für die erforderlichen Sicherheitsmanagement-Aspekte gibt. Dies gilt insbesondere für Beauftragung, Projektleitung und Qualitätssicherung von Software-Entwicklungen, reicht aber auch bis zu Compliance- und Risikomanagement.

Ich freue mich daher sehr, dass Matthias Rohr ein Grundlagenbuch zu diesem Thema geschrieben hat. Ein wesentliches Merkmal dieses Buches ist, dass es die verschiedenen Typen von Schwachstellen, Angriffen und Gegenmaßnahmen strukturiert und damit eine Vollständigkeit erreicht, die ich bisher in derartigen Büchern nur selten gesehen habe. Das Werk geht ganz wesentlich auf die Management- und Steuerungsaspekte ein, ist leicht zu lesen und auch für Nicht-Sicherheits-Spezialisten sehr gut verdaulich. Ich hoffe, dass dieses Buch den Weg auf viele Schreibtische findet und damit hilft, die moderne Welt der IT eine Spur sicherer zu machen.

Brandenburg, im Mai 2014 Prof. Dr. Sachar Paulus

Vorwort des Autors

Dieses Buch basiert auf den Praxiserfahrungen, die ich in den letzten zehn Jahren aus unzähligen Projekten, Unternehmen und Gesprächen gesammelt habe. Ich habe dieses Buch geschrieben, um meine Erfahrungen weiterzugeben und eine bislang fehlende Gesamtsicht auf das Thema Webanwendungssicherheit, speziell im Hinblick auf deren unternehmerischen Einsatz, darzustellen. Zudem soll dieses Buch eine Orientierungshilfe durch den Dschungel an Begriffen, Konzepten und Verfahren sein, die in diesem Bereich existieren und nicht nur Neulinge schnell verwirren können.

Mit diesem Buch sollen nicht nur Experten angesprochen werden, deren alltägliche Herausforderungen bei der Entwicklung von Webanwendung genau hierin bestehen, sondern auch solche Personen, die sich bislang noch nicht stärker mit dem Thema befasst haben. Die einzelnen Kapitel richten sich nicht nur an unterschiedliche Lesergruppen, sondern setzen auch unterschiedliche Vorkenntnisse voraus:

- **Kapitel 2** beschreibt die Hintergründe von unsicheren Webanwendungen und Grundlagen der Webanwendungssicherheit. Der erste Teil dieses Kapitels kann von Lesern mit entsprechenden Vorkenntnissen übersprungen werden.
- **Kapitel 3** stellt die wichtigsten Angriffe und Schwachstellen von Webanwendungen dar. Wer sich hiermit bereits eingehend beschäftigt hat (z. B. erfahrene Security Consultants), dem empfehle ich die einzelnen Abschnitte zumindest im Hinblick auf neue Aspekte zu überfliegen und sich die verwendete Taxonomie zu vergegenwärtigen.
- **Kapitel 4** widmet sich ausführlich den Maßnahmen, mit denen sich Webanwendungen im Hinblick auf gängige Bedrohungen absichern lassen. Dieses Kapitel wendet sich an Personen, die an der Webentwicklung beteiligt sind (also vor allem Entwickler) sowie an Tester, die entsprechende Empfehlungen und Prüfungen ableiten. Wer nicht zu tief in die Materie einsteigen will, dem erlauben die Zusammenfassungen der einzelnen Unterkapitel die wichtigsten Aspekte zu erfassen. Zusätzlich werden zentrale Aussagen über Auszeichnungsblöcke herausgestellt.
- **Kapitel 5** liefert einen Überblick über relevante Bewertungs- und Prüfverfahren innerhalb der einzelnen Entwicklungsphasen und bietet für jeden Leser, der sich mit Sicherheitsanalysen oder -Tests von Webanwendungen beschäftigt, interessante Aspekte.

- **Kapitel 6** rundet die Betrachtung durch die Beschreibung organisatorischer Aspekte der Anwendungssicherheit ab und wendet sich vor allem an Projekt- oder Entwicklungsleiter sowie das IT-Security- und Qualitäts-Management.

Dieses Buch verfolgt nicht den Anspruch, alle möglichen, denkbaren und zukünftigen Betrachtungsweisen und Spezialitäten der Sicherheit von Webanwendungen vollständig zu beschreiben. Die Disziplin der Webanwendungssicherheit ist derart vielseitig und vielschichtig, dass hierfür ein Buch sicher nicht ausreichen würde. Daher werde ich an vielen Stellen nicht auf alle bekannten Aspekte eingehen oder alle denkbaren Angriffe und Schwachstellen für Webanwendungen darstellen können, sondern auf weiterführende Literatur verweisen.

Das Buch hätte niemals in der vorliegenden Form entstehen können, wenn ich nicht durch viele Menschen unterstützt worden wäre. Für die aktive Mitarbeit an Inhalten und der Gliederung danke ich dabei im Besonderen Thomas Schreiber von der SecureNet GmbH. Zudem bedanke ich mich auch sehr herzlich bei Herrn Professor Paulus, der so freundlich war, ein bestechendes Geleitwort zu schreiben. Weiterhin möchte ich mich für die großartige Unterstützung von Markus Miedaner, Thomas Skora, Markus Kulicke, Christian Schneider, Arndt Allmeling, David Matscheko, Thomas Kerbl, Max Herold, Andreas Kalender, Tim Eggert herzlichst bedanken. Meinem Vater, Wolfgang Rohr, danke ich ganz herzlich für viele Korrekturhinweise.

Die in diesem Buch zusammengestellten Aussagen und Empfehlungen stellen den bei Drucklegung aktuellen Stand dar. Bedingt durch die technologische Entwicklungen und das Bekanntwerden neuer Bedrohungen können einzelne möglicherweise im Laufe der Zeit an Gültigkeit oder Vollständigkeit verlieren.

Für entsprechende Hinweise und auch Anregungen aller Art bin ich überaus dankbar! Sofern zutreffend werde ich versuchen, diese in einer zukünftigen Auflage zu berücksichtigen. Bitte besuchen Sie hierzu die Webseite des Buches: www.webappsecbuch.de. Dort können Sie weitere Informationen sowie einen exemplarischen Sicherheitsstandard für Webanwendungen finden. Vielen Dank!

Hamburg, im September 2014 Matthias Rohr

Inhaltsverzeichnis

Einleitung 1

Der Programmierwahn von heute ist das Security-Desaster von Morgen" Brian Chess, Fortify Inc.

Zusammenfassung

In diesem Kapitel wird einleitend auf zwei Themengebiete eingegangen. Zunächst werden dabei verschiedene Aspekte dargestellt, die erklären, warum derart viele Sicherheitsprobleme in heutigen Webanwendungen vorkommen. Im zweiten Teil wird auf den eigentlichen Begriff „Webanwendungssicherheit" genauer eingegangen und dargestellt, wie sich diese Disziplin in die IT-Sicherheit einfügt.

1.1 Zum Begriff „Webanwendung"

Bevor auf einzelne Sicherheitsthemen eingegangen wird, muss zunächst geklärt werden, was in diesem Zusammenhang unter einer „Webanwendung" konkret zu verstehen ist. Eine Webanwendung muss dabei keinesfalls über das Web erreichbar sein, wie der Name suggerieren mag. Webanwendungen werden heutzutage im großen Maße auch innerhalb von Unternehmen eingesetzt, etwa für die Abbildung des Intranets. Entscheidend dafür, dass wir eine Anwendung als Webanwendung bezeichnen, ist einzig der technologische Aspekt, nämlich der Einsatz von Webtechnologien. Als Grundlage soll daher die folgende, recht einfache Definition dienen:

► **Webanwendung** Eine Webanwendung ist eine Client-Server-Anwendung, die auf Webtechnologien (HTTP, HTML etc.) aufsetzt.

M. Rohr, *Sicherheit von Webanwendungen in der Praxis,* Edition <kes>,
DOI 10.1007/978-3-658-03851-9_1

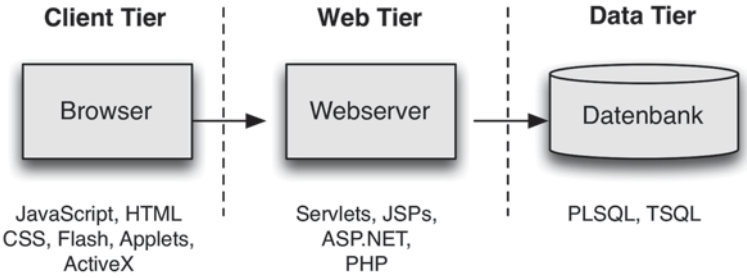

Abb. 1.1 3-Tier-Architektur einer Webanwendung

Eine Webanwendung wird dabei meistens über einen Webbrowser (kurz Browser) aufgerufen, in dem der serverseitig generierte HTML-, JavaScript- und CSS-Code ausgewertet („geparst“) und dargestellt wird. Da wir neben einem Browser auch auf andere Weise auf eine Webanwendung zugreifen können (z. B. mit einem Testtool von der Kommandozeile aus), wird oftmals allgemeiner von einem User Agent gesprochen. Zur Kommunikation zwischen Browser (also User Agent) und Server dient dabei vor allem das HTTP-Protokoll.

Serverseitig werden Webanwendungen auf Web- und Applikationsservern[1] ausgeführt, die dann wiederum in der Regel auf Hintergrundsysteme, z. B. eine Datenbank, zugreifen. Daraus ergibt sich eine sogenannte 3-Tier-Architektur (dreischichtige Architektur), die in Abbildung Abb. 1.1 dargestellt ist. Zur Veranschaulichung wurden dabei exemplarische Technologien den jeweiligen Schichten zugeordnet.

Für die Darstellung einer modernen Enterprise-Webanwendung reicht eine solche 3-Tier-Architektur allerdings kaum mehr aus. Stattdessen finden wir dort in der Regel deutlich mehr als drei Schichten vor. So wird häufig nicht nur ein, sondern gleich eine ganze Vielzahl von Hintergrundsystemen (etwa SAP-Systeme, ESBs etc.) angebunden, wozu wiederum unterschiedliche Technologien (allen voran XML-basierte Webservices) verwendet werden. Der serverseitige Code wird dabei auf Anwendungsservern (z. B. WebSphere) ausgeführt. Vorab übernimmt ein separater Webserver (z. B. Apache) die eigentliche HTTP-Kommunikation.

Auch auf der Clientseite hat sich in den vergangenen Jahren einiges getan. Neben dem klassischen Webbrowser werden Webanwendungen auch von Smartphones und Tablets aufgerufen. Selbst native Apps, die nicht in JavaScript und HTML, sondern in Objective-C (iOS) oder Java (Android) geschrieben sind, greifen vielfach auf externe Schnittstellen von Webanwendungen zu und arbeiten damit als Webclients. Neben HTTP kommen dabei Protokolle wie JSON, XML-RPC sowie zukünftig auch WebSockets (bzw. RFC 6455) zum Einsatz.

[1] Im Rahmen dieses Buches wird aus Gründen der Vereinfachung meist nur von „einem Webserver“ gesprochen, auch wenn damit in der Praxis diverse Cluster von Web- und Applikationsservern gemeint sein können, die aus vielen hunderten oder sogar tausenden Systemen bestehen.

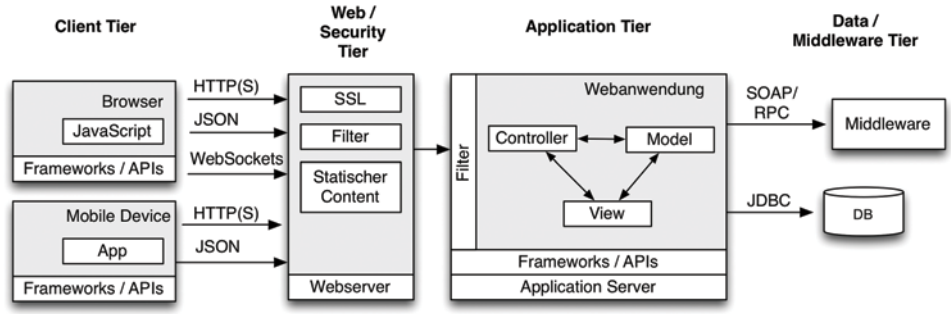

Abb. 1.2 N-Tier-Architektur einer Enterprise-Webanwendung

Architektonisch basieren die meisten Webanwendungen auf dem MVC Pattern, das die Darstellung (View) von der Geschäftslogik und Datenschicht (Model) sowie von der Steuerung (Controller) entkoppelt. Auf jeder dieser Ebenen lassen sich eine Vielzahl weiterer APIs und Frameworks wie z. B. Template-Technologien in der View oder ORM-Frameworks im Model verwenden. Daraus ergibt sich ein deutlich komplexeres Bild von dem, was wir heutzutage als Webanwendung vorfinden und selbst dies ist noch stark vereinfacht. Abbildung 1.2 zeigt ein Beispiel einer solchen N-Tier-Architektur:

1.2 Warum sind Webanwendungen unsicher?

Die Gründe warum moderne Webanwendungen so viele Sicherheitsmängel enthalten sind vielschichtig und umfassen viele aber nicht ausschließlich technische Ursachen. Im Folgenden sollen die wichtigsten hiervon beläuchtet werden.

1.2.1 Anwendungslayer versus Netzwerklayer

Unternehmen setzen vielfach IT-Sicherheitsmaßnahmen über Infrastrukturkomponenten wie Netzwerkfirewalls oder IDS-/IPS-Systeme um. Das Problem dabei ist, dass diese überwiegend auf der Ebene des TCP/IP-Protokolls arbeiten und damit für die Abwehr von Angriffen, die über höhere Protokolle des Anwendungslayers (z. B. dem HTTP-Protokoll) durchgeführt werden, nur bedingt geeignet sind. Denn sie verstehen diese Protokolle nicht, oder wenn, dann nur sehr eingeschränkt. Das ist in etwa mit einem Zöllner zu vergleichen, der nur die Papiere eines Containers kontrollieren kann, nicht jedoch in diesen selbst hineinschauen darf (Abb. 1.3).

Genauso unterscheiden wir Angriffe auf dem Netzwerklayer (z. B. Netzwerk Spoofing) von solchen auf dem Anwendungslayer, die im Kontext von Webanwendungen auch als Webangriffe bezeichnet werden. Gerade in größeren Unternehmen wird durch die Netzwerkinfrastruktur ein relativ hohes Sicherheitsniveau auf der Ebene des Netzwerklayers gewährleistet. Ganz anders sieht es auf dem Anwendungslayer aus. Daher ist es auch kaum verwunderlich,

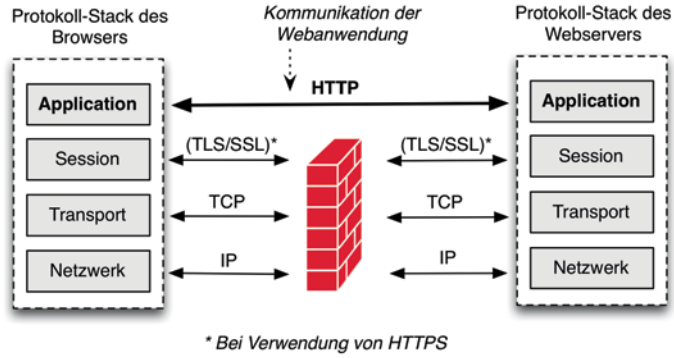

Abb. 1.3 Netzwerk- vs. Anwendungslayer

dass Cyber-Angriffe heutzutage in den überwiegenden Fällen darüber ausgeführt werden – meistens gegen Webanwendungen. So neu ist diese Situation dabei keinesfalls: Bereits im Jahr 2003 ging die US-Standardisierungsbehörde NIST davon aus, dass drei von vier Angriffen auf IT-Systeme auf der Ebene des Anwendungslayers erfolgen. Laut UK Breach Report lag dieser Anteil im Jahr 2010 bereits bei 86 % (vergl. [1]).

Auch im Rahmen von verschiedenen Sicherheitsuntersuchungen unterscheiden wir bewusst zwischen solchen, die sich nur auf den Netzwerklayer beziehen und solchen, die anwendungsseitig durchgeführt werden (z. B. anwendungsseitige Pentests gegen eine Webanwendung).

► Angriffe gegen Webanwendungen werden auf dem Anwendungslayer –überwiegend mittels des HTTP-Protokolls – durchgeführt. Netzwerkseitige Sicherheitsmaßnahmen beziehen sich auf die Ebenen darunter und können Angriffe gegen eine Webanwendung daher in der Regel weder erkennen noch verhindern.

Dennoch behandeln viele IT-Sicherheits-Standards, wie etwa der IT-Grundschutz, immer noch fast ausschließlich rein infrastrukturelle Sicherheitsaspekte. In konkrete Maßnahmen zur Steigerung der Anwendungssicherheit fließt gewöhnlich nur ein geringer Teil des IT-Sicherheitsbudgets; der überwiegende Teil wird nach wie vor in den infrastrukturellen Bereich investiert (vergl. [2]). Allerdings wird dort mittlerweile mit Hilfe von Next Generation Firewalls der Anwendungslayer ebenfalls zunehmend abgedeckt.

1.2.2 Design-Entscheidungen des HTTP-Protokolls

Wer sich mit Webanwendungen bzw. Webtechnologien beschäftigt, kommt am HTTP-Protokoll nicht vorbei. HTTP ist das diesen Technologien zugrundeliegende Übertragungsprotokoll[2]. Spezifiziert wurde es bereits im Jahr 1981 und ist damit noch deutlich

[2] Mit WebSockets existiert seit kurzem eine weitere Übertragungs-Technologie im Web, welche ein eigenes Protokoll (RFC 6455) einsetzt. Für die Zukunft ist davon auszugehen, dass dieses Protokoll zusätzlich zu HTTP deutlich häufiger zum Einsatz kommt.

älter als das Web selbst (die Spezifikation findet sich in RFC 793). Wozu das Protokoll einmal verwendet wird, konnte den Schöpfern von HTTP damals nicht wirklich bewusst sein. Vielmehr bestand das Ziel in der Spezifikation von HTTP in der Schaffung eines einfachen und robusten Client-Server-Protokolls, über das heterogene Systeme leicht miteinander kommunizieren konnten. Denn TCP-Protokolle wie HTTP besaßen vor allem zwei Designziele: Interoperabilität und Robustheit:

> TCP-Implementierungen werden einem generellen Robustheits-Prinzips folgen: Sei konservativ in dem, was du tust und liberal, in dem was du von anderen akzeptierst. Jon Postel, RFC 793, Sept. 1981

Sicherheit hingegen stand damals sicherlich nicht im Fokus der Autoren von HTTP. Die zunächst spezifizierte Protokollversion 0.9 wurde im Laufe der Jahre noch zweimal überarbeitet (HTTP 1.0 und HTTP 1.1) und auch vereinzelt um Sicherheitsaspekte erweitert. Von den grundlegenden Design-Entscheidungen wurde dabei jedoch nicht mehr abgewichen. So lässt sich auch mit HTTP 1.1 (spezifiziert in RFC 2616) noch auf sehr einfache Weise auf eine Ressource im Web zugreifen. Prinzipiell benötigen wir hierzu nur zwei Zeilen:

```
GET /welcome.html?param1=1&param2=2 HTTP/1.1
Host: www.example.com
```

Der Server antwortet auf diesen HTTP Request mit einer ebenfalls sehr einfachen Antwort (der „HTTP Response“). Die für den Browser wichtigste Information steht gleich in der ersten Zeile des HTTP-Headers, nämlich der HTTP Status Code. Über ihn gibt der Webserver an, ob eine Anfrage erfolgreich war (Status Code 200), eine angefragte Ressource nicht existiert (404), eine Authentifizierung erforderlich ist (403) oder der Browser eine Weiterleitung durchführen soll (303). War die Anfrage erfolgreich, hängt der Webserver die angeforderte Ressource (also etwa den HTML-Code einer aufgerufenen Webseite) im unteren Teil der Antwort (HTTP Body) an:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 3434
Date: Tue, 36 Mar 2013 10:50:10 GMT

<html>
[HTML-Markup]
…
```

In diesem Fall handelt es sich um eine HTML-Seite, was durch „Content-Type: text/html“ angegeben wird. Zusätzlich wird mit „Content-Length“ die Größe des Inhaltes in Bytes und mit „Date“ sein letztes Änderungsdatum angegeben. Die Einfachheit dieser Spezi-

fikation von HTTP besitzt viele Vorteile und zugegebenermaßen auch einigen Charme. In Bezug auf die Sicherheit von Webanwendungen ergeben sich daraus allerdings gleich mehrere Probleme:

HTTP-Aufrufe lassen sich über URLs parametrisieren

Der Aufruf aus dem obigen Beispiel lässt sich sehr einfach generieren und zwar durch einen Browser. In diesem Fall müssten wir dort nämlich nur die folgende URL eintippen:

```
http://www.example.com/welcome.html?param1=1&param2=2
```

Der Browser kümmert sich daraufhin um die Generierung der entsprechenden Anfrage – wobei er gewöhnlich noch eine Reihe zusätzlicher Header an den HTTP Request anhängt. In einer URL lassen sich natürlich auch Parameter angeben. Eingeleitet werden diese durch ein „?“. Der Teil der URL, der darauf folgt, wird „Query String“ genannt. Da Query Strings auf der Ebene des HTTP-Protokolls per HTTP-GET-Methode übertragen werden, sprechen wir auch von GET-Parametern.

Die andere zentrale HTTP-Methode ist HTTP POST, wobei die Parameter nicht mehr in der URL, sondern im Request Body übertragen werden und damit nicht mehr in der Adresszeile des Browsers sichtbar sind. Für die Übertragung von Formularinhalten ist dies gewöhnlich das Standardverfahren. Eine entsprechende Anfrage hat die folgende Gestalt:

```
POST /welcome.html HTTP/1.1
Host: www.example.com
Content-Length: 17

param1=1&param2=2
```

Allerdings unterstützen viele Webframeworks beide HTTP-Methoden transparent. Dadurch lassen sich entsprechende POST-Anfragen häufig auch per HTTP GET, also als URL, an die Anwendung senden. Die Einfachheit in dieser HTTP-Parametrisierung spielt natürlich auch potentiellen Angreifern in die Hände, indem sie sich die Eigenschaft zunutze machen, dass einem bei einer Webseite angemeldeten Benutzer auf diese Weise eine parametrisierte URL untergeschoben werden kann. Dieses Vorgehen wird als indirekter Angriff (Abschn. 2.1.3) bezeichnet. Zu diesen zählt neben Cross-Site Scripting (Abschn. 2.6.3) auch Cross-Site Request Forgery (Abschn. 2.6.4) und damit gleich zwei der häufigsten Sicherheitsprobleme von Webanwendungen.

HTTP ist unverschlüsselt

Mit HTTPS (Abschn. 3.5.3) existiert bereits seit 1994 ein Protokoll, durch das HTTP um einen zusätzlichen SSL/TLS-Layer erweitert wird. Dieser sorgt dafür, dass sämtliche übertragene Daten nicht nur verschlüsselt werden, sondern beide Seiten sich auch mittels SSL-Zertifikaten (genauer X.509-Zertifikaten, Abschn. 3.5.2) gegenseitig authentifizie-

ren können. Allerdings ist die Verwendung von HTTPS (mal von Online-Banking und ähnlichem abgesehen) keinesfalls die Norm, so dass der Großteil der Webkommunikation heute unverschlüsselt durchgeführt wird. Das ermöglicht es Angreifern zum einen, sensible Daten abzufangen und zu manipulieren. Zum anderen können sich die beiden Seiten (also Browser und Webserver) hierdurch gar nicht miteinander auf sichere Weise authentisieren. Ohne HTTPS authentisiert sich ein Server bei dem Client (bzw. Browser) daher in der Regel ausschließlich über seinen DNS-Namen. Gelingt es einem Angreifer, den DNS-Server eines Unternehmens zu übernehmen, kann er dadurch auch alle seine Webseitenzuweisungen kontrollieren. Auch die eigene IP-Adresse lässt sich von einem Angreifer in bestimmten Umgebungen fälschen („spoofen“).

HTTP besitzt keinen Integritätsschutz

Genauso wie die HTTP-Übertragung unverschlüsselt ist und damit von einem Dritten (wir sprechen hier auch von einem „Man-in-the-Middle“, Abschn. 1.2.3) mitgelesen werden kann, ist sie beliebig manipulierbar, da sie keinerlei Integritätsschutz bietet. Sofern die Übertragung auch in diesem Fall nicht explizit über HTTPS durchgeführt wird, wodurch ein entsprechender Schutz geboten wäre, lässt sich die HTTP-Übertragung in beiden Richtungen des Übertragungsweges manipulieren, ohne dass dies zu Fehlern führen würde.

HTTP ist zustandslos

Auch Zustände kennt HTTP nicht. Stattdessen operiert es völlig losgelöst vom darunter liegenden TCP-Stack, also etwa einer persistenten Netzwerkverbindung. Jeder HTTP Request ist daher von jedem vorangegangenen komplett unabhängig. Solche Zustände, wie etwa die Tatsache, dass der Client sich mit einem vorherigen Request bereits erfolgreich authentisiert hat und nun auf einem bestimmten Benutzerkonto operiert, müssen zwischen Client und Server separat aufrechterhalten werden. In der Praxis geschieht dies dadurch, dass der Server jedem Request einen eindeutigen Identifier (Session-ID) zuordnet, der dann dem Client mitgeteilt wird. Für die Abbildung eines solchen (HTTP) Session Managements dienen vor allem zwei Verfahren: HTTP-Cookies (Session-Cookies) und URL Rewriting.

Im erstgenannten Fall sendet die Anwendung (bzw. der Webserver) nach erfolgreicher Authentifizierung einen HTTP-Cookie, der die Session-ID erhält, an den Browser. Praktisch handelt es sich dabei lediglich um einen speziellen HTTP Response Header, der die folgende Form hat:

```
HTTP/1.1 200 OK
...
Set-Cookie: SESSIONID=1234567
```

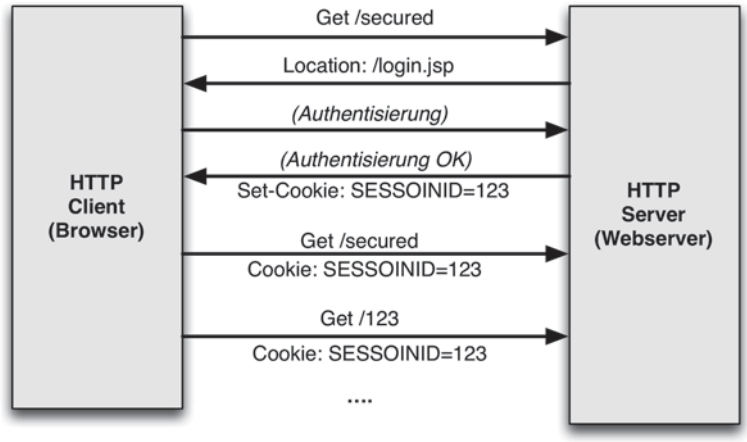

Abb. 1.4 Funktionsweise eines Cookie-basierten Session Managements

Nur durch Erhalt dieses Headers weiß der Browser, dass er die dort angegebene Session-ID (die hier aus Gründen der Veranschaulichung deutlich vereinfacht wurde) nun künftig mit jeder Anfrage an diesen Host mitsenden muss, wofür er wiederum einen entsprechenden Request Header verwendet:

```
GET /secured HTTP/1.1
Host: www.example.com
...
Cookie: SESSIONID=1234567
```

Nachdem sich ein Benutzer an einer Webseite angemeldet hat, authentifiziert der Server jeden weiteren Request ausschließlich über die Session-ID und ordnet diese einer authentifizierten Sitzung (bzw. einem konkreten Benutzerkonto) zu. Der vollständige Ablauf eines solchen Cookie-basierten Session Managements ist in Abb. 1.4 exemplarisch dargestellt:

Ein cookie-basiertes Session Management hat zur Folge, dass angemeldeten Benutzern leicht Anfragen von einem Dritten untergeschoben werden können. Der entsprechende Angriff, nämlich Cross-Site Request Forgery (Abschn. 2.6.4), wurde bereits angesprochen. Mit URL Rewriting existiert ein alternatives Session-Management-Verfahren, bei dem diese Angreifbarkeit erst einmal nicht besteht. Statt in Cookies werden die Session-IDs dabei automatisch vom Webserver in alle lokalen URLs eingebaut. Das sieht dann etwa wie folgt aus:

```
https://www.example.com/action/account/show;sessionid=1234567?param1=1
```

Allerdings besitzt dieses Verfahren eine ganze Reihe zusätzlicher Sicherheitsproblematiken. Zum Beispiel kann die in der URL transportierte Session-ID auf verschiedenste Weise offengelegt werden: in Logdateien, in kopierten HTML-Seiten (per “Copy-and-Pa-

ste“)[3] oder in HTTP Referern. Über letztere teilt ein Browser einem Webserver innerhalb des HTTP Headers mit, von welcher Seite (bzw. URL) ein Benutzer auf die Seite geleitet wurde. Klickt ein Benutzer auf der obigen Seite etwa auf einen externen Link, so würde der folgende Referer an diese Seite übermittelt:[4]

```
GET /link HTTP/1.1
Host: www.example.net
Referer: https://www.example.com/account/show;sessionid=1234567?param1=1
```

- In der Regel stellt die Session-ID das einzige Merkmal dar, mit dem ein Server eine HTTP-Anfrage einer vorherigen zuordnen kann, um sie mit einem angemeldeten Benutzer und einer entsprechenden Sitzung zu assoziieren. Gelangt ein Angreifer in den Besitz einer gültigen Session-ID, so kann er mit dieser in den meisten Fällen auch auf das entsprechende Benutzerkonto zugreifen.

Unsichere Authentifizierungsverfahren

Das HTTP-Protokoll beschreibt zwar mit HTTP Basic und HTTP Digest verschiedene Authentifizierungsverfahren, allerdings besitzen diese gravierende Sicherheitsprobleme und sind zudem in moderne Webanwendungen nicht integrierbar. Die Folge ist, dass Webanwendungen fast immer ein eigenes Verfahren zur Authentifizierung von Benutzern implementieren müssen, was als „HTTP Forms“ lediglich eine offizielle Bezeichnung, jedoch keinerlei standardisierte Sicherheitsaspekte[5], besitzt. Da es für die Implementierung einer solchen formularbasierten Authentifizierung somit an konkreten Vorgaben fehlt, ist es kaum verwunderlich, dass in der Praxis viele Webanwendungen in diesem Bereich teilweise erhebliche Sicherheitsmängel aufweisen.

1.2.3 Man-in-the-Middle-Proxys und Browser-Plugins

Webanwendungen sind auch deshalb so leicht angreifbar, weil ein Angreifer sämtliche, zwischen Client und Server übertragene Daten, beliebig manipulieren kann. Auch solche,

[3] In diesem Fall kann ein Benutzer seine eigene Session dadurch kompromittieren, dass er im angemeldeten Bereich Inhalte kopiert und per Mail versendet. Im so kopierten HTML-Markup sind dann die Links mit der jeweiligen Session-ID des Benutzers enthalten. Klickt der Empfänger dieser E-Mail nun auf einen der Links, so wird er über die darin enthaltene Session-ID an der Anwendung angemeldet (vergl. Session Hijacking, Abschn. 2.6.6).

[4] Natürlich wird nicht bei jedem Zugriff ein Referer-Header mitgesendet. Etwa dann nicht, wenn der Zugriff über einen Bookmark, JavaScript oder durch Eingabe der Adresse in die Adresszeile erfolgte. Auch wird ein Referer nur dann übermittelt, wenn die aufgerufene Seite das gleiche Schema (also z. B. „https://“) besitzt.

[5] Allerdings existieren hierfür mittlerweile verschiedene standardisierte Implementierungen. So z. B. JAAS im Fall von Java-basierten Webanwendungen, das von den meisten Applikationsservern unterstützt wird und in aktuellen Versionen inzwischen auch verschiedene Sicherheitsaspekte abbildet.

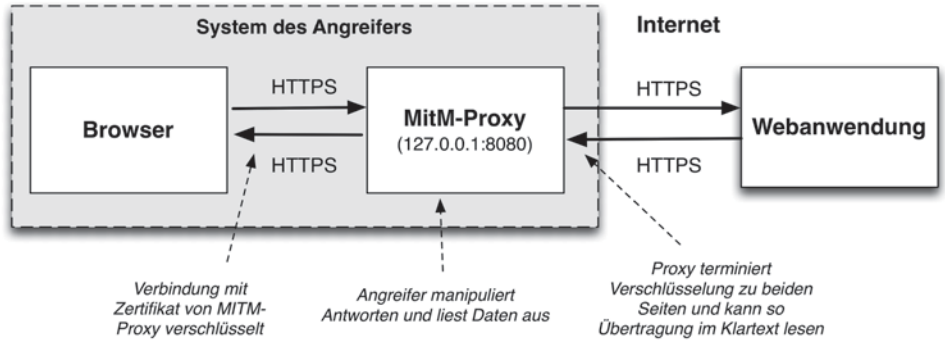

Abb. 1.5 Funktionsweise eines MitM-Proxys

die über kein Formular eingegeben werden, sondern etwa über sogenannte Hidden Fields (vgl. Beispiel in Abschn. 1.2.6) versteckt ins Webformular gesetzt und im Hintergrund mit den übrigen Formulardaten übertragen werden.

Möglich ist dies vor allem, da Browser eine – freilich für einen anderen Zweck vorgesehene – Funktion bieten, nämlich die Verwendung eines HTTP-Proxys. Auf diese Weise lässt sich ein lokales Proxy-Tool einbinden, über das die gesamte Kommunikation zwischen Browser und Webanwendung geleitet wird und sich darin beliebig auslesen und manipulieren lässt. Da diese Tools hier als sogenannter „Man-in-the-Middle" agieren, verwenden wir für diese Toolgattung auch den Begriff „Man-in-the-Middle-Proxy" (MitM-Proxy). Abbildung 1.5 veranschaulicht die Funktionsweise eines solchen Proxys.

Der MitM-Proxy lauscht hier auf dem lokalen Interface des Angreifers (127.0.0.1), und zwar in diesem Fall auf Port 8080. Über eine entsprechende Einstellung im Browser leitet dieser nun sämtliche Kommunikation über den MitM-Proxy (siehe Abb. 1.6):

Der Screenshot in Abb. 1.6 zeigt den häufig eingesetzten Burp-Proxy, in dem gerade ein vom Browser gesendetes Formular abgefangen wurde und sich nun manipulieren lässt, bevor die modifizierte Anfrage durch Klicken auf „Forward" an den Webserver weiterleitet wird (Abb. 1.7).

Ein solcher MitM-Proxy stellt für Tester und Angreifer gleichermaßen das Universalwerkzeug zum Aufspüren von Schwachstellen in Webanwendungen dar. Es legt die Eintrittspunkte der serverseitigen Anwendung komplett offen und gibt dem Angreifer über einen großen Funktionsumfang weitreichende Möglichkeiten, auf die Anwendung frei von jeglichen Browser-Restriktionen (z. B. clientseitige Validierung) mit beliebigen Daten einzuwirken und gezielte wie automatisierte Tests und Angriffe durchzuführen.

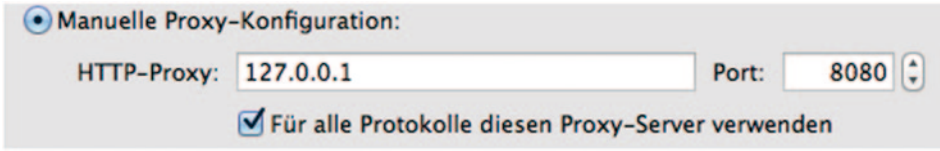

Abb. 1.6 Einbinden eines MitM-Proxys im Browser

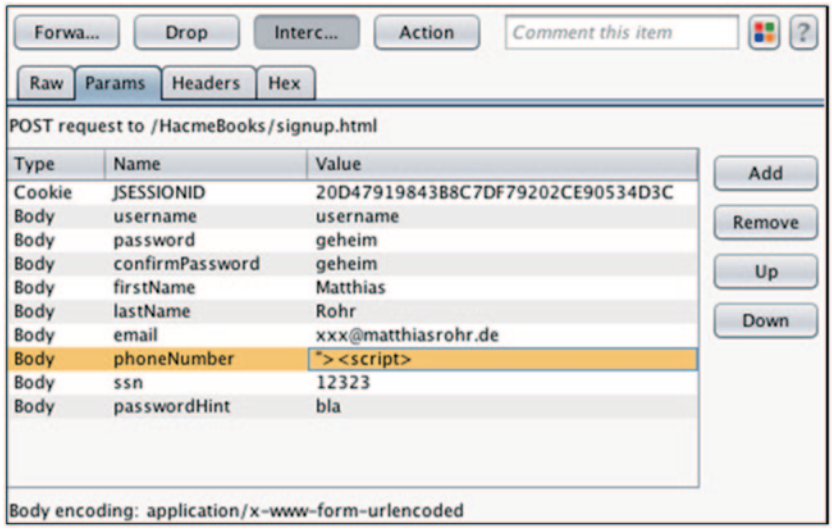

Abb. 1.7 Burp Proxy

Dies betrifft natürlich auch verschlüsselte HTTPS-Verbindungen, da sie ebenfalls einfach am MitM-Proxy terminiert werden und von dort wiederum per HTTPS zum Server oder Client übermittelt werden. Dies führt dann zwar im Browser zu einem Zertifikatsfehler (schließlich ist dieses ungültig), was einen Angreifer in der Regel aber nicht wirklich stört. Auch die Kommunikation von Mobile Apps lässt sich auf diese Weise offenlegen und manipulieren. Denn auch gängige Smartphone-Plattformen bieten die Möglichkeit des Einstellens eines HTTP-Proxys.

Doch das Arsenal eines Angreifers ist keinesfalls nur auf den Einsatz solcher MitM-Proxys beschränkt. Insbesondere für Firefox und Chrome existiert eine Vielzahl entsprechender Browser-Erweiterungen (die meist unter „Developer Tools“ kategorisiert sind), mit denen sich praktisch sämtliche clientseitigen Bestandteile einer Webanwendung beliebig manipulieren lassen. Neben dem HTML- und JavaScript-Code natürlich auch den clientseitigen Storage (Cookies, HTML5 Web Storage, etc.): Abb. 1.8 zeigt, wie dies innerhalb des Chrome Browsers möglich ist.

► Ein Angreifer kann die gesamte Kommunikation sowie die clientseitige Anwendungslogik und dort gespeicherte Werte mit entsprechenden Tools beliebig manipulieren und dadurch sämtliche Browser-Restriktionen aushebeln.

1.2.4 Unzureichende Validierung von Eingabedaten

Webanwendungen dienen in erster Linie der Verarbeitung von Daten – und in vielen Fällen stammen diese in Form von HTTP-Parametern von Benutzern. So ist es kaum verwunderlich, dass ein großer Teil der Angriffe auf Webanwendungen auf die Manipulation dieser Parameter zurückzuführen ist. Die Manipulation eines HTTP Requests ist dabei