

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

© Springer Fachmedien Wiesbaden 2015

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften. Der Verlag, die Autoren und die Herausgeber gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag noch die Autoren oder die Herausgeber übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen.

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 Gesamt-sicht 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 Auszeichnungsblocke 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

- 1 Einleitung** 1
 - 1.1 Zum Begriff „Webanwendung“ 1
 - 1.2 Warum sind Webanwendungen unsicher? 3
 - 1.2.1 Anwendungslayer versus Netzwerklayer 3
 - 1.2.2 Design-Entscheidungen des HTTP-Protokolls 4
 - 1.2.3 Man-in-the-Middle-Proxys und Browser-Plugins 9
 - 1.2.4 Unzureichende Validierung von Eingabedaten 11
 - 1.2.5 Webanwendungen sind häufig Mehrbenutzersysteme 13
 - 1.2.6 Offenlegung serverseitiger Objekte und Funktionen 13
 - 1.2.7 Einkodierung 15
 - 1.2.8 JavaScript 17
 - 1.2.9 Fehlende Kontrolle der Ausführungsumgebung 18
 - 1.2.10 Hochprivilegierter Programmcode 19
 - 1.2.11 Unzureichende Absicherung von Backendsystemen 19
 - 1.2.12 Unsichere 3rd-Party-Komponenten und Legacy Code 20
 - 1.2.13 Jeder kann heute eine Webanwendung zusammenbauen 22
 - 1.2.14 Die Agilität moderner Webentwicklung 22
 - 1.2.15 Die Gefahr wird unterschätzt 24
 - 1.2.16 Eingeschränkte Sichtbarkeit und Zielkonflikte 26
 - 1.3 Was ist Webanwendungssicherheit? 28
 - 1.3.1 Begriffsbestimmung 28
 - 1.3.2 Nicht das gleiche wie „sichere“ Webanwendungen 30
 - 1.3.3 Etwas anderes als Websicherheit 31
 - 1.3.4 Das Management von Risiken 31
 - 1.3.5 Der Schutz personenbezogener Daten 33
 - 1.3.6 Der Umgang mit Vertrauen 34
 - 1.3.7 Softwarequalität 35
 - 1.3.8 Nicht durch Technik alleine zu lösen 36
 - 1.3.9 Ein Querschnittsthema 36
 - 1.3.10 Ein evolutionärer Prozess 38

1.4	Relevante Organisationen	39
1.5	Zusammenfassung	41
2	Bedrohungen für Webanwendungen	43
2.1	Begriffe und Konzepte	43
2.1.1	Bedrohung, Bedrohungsquelle und Gefährdung	44
2.1.2	Schwachstellen und Sicherheitslücken	45
2.1.3	Angriffe	49
2.2	Relevante Standards und Projekte	54
2.3	Manipulation der Anwendungslogik	56
2.4	Pufferüberläufe (Buffer Overflows)	58
2.5	Interpreter Injection	59
2.5.1	SQL Injection	60
2.5.2	OS Command Injection	64
2.5.3	Serverseitiges Code Injection	65
2.5.4	XML Injection	67
2.6	Clientseitige Angriffe	69
2.6.1	Hintergrund: Die Same Origin Policy (SOP)	69
2.6.2	Hintergrund: Ajax-Zugriffe	71
2.6.3	Cross-Site Scripting (XSS)	72
2.6.4	Cross-Site Request Forgery (CSRF)	80
2.6.5	Cross-Site Redirection	83
2.6.6	Sitzungsfixierung (Session Fixation)	85
2.6.7	Session Hijacking	86
2.6.8	Website Spoofing & Defacement	89
2.6.9	Clickjacking	91
2.6.10	Drive by Infection („Malwareschleudern“)	92
2.7	Angriffe auf Benutzerkonten und Privilegien	94
2.7.1	Ermitteln von Passwörtern (Brute Forcing etc.)	94
2.7.2	Passwort Recycling	96
2.7.3	Ungeschützte Ressourcen	96
2.7.4	Path Traversal	98
2.7.5	Privilegienerweiterung	99
2.7.6	Überprivilegierung	101
2.7.7	Hintertüren (Backdoors)	102
2.8	Information Disclosure	103
2.9	Unbeabsichtigte Aktionen	105
2.9.1	Race Conditions	105
2.9.2	Replay und „Verklicken“	107
2.10	Denial of Service (DoS)	108
2.11	Zusammenfassung	109

3 Technische Sicherheitsmaßnahmen	113
3.1 Begriffe und Konzepte	114
3.1.1 Sicherheitsmechanismen (Security Controls)	114
3.1.2 Sicherheitsanforderungen	116
3.1.3 Quick Wins und Best Practices	117
3.1.4 Angemessene Sicherheit	117
3.2 Relevante Standards und Projekte	120
3.2.1 Bundesdatenschutzgesetz (BDSG)	120
3.2.2 PCI-DSS	121
3.2.3 Secure Coding Guidelines	122
3.2.4 BSI Grundschriftkataloge und Studien	122
3.2.5 NIST Special Publications	123
3.2.6 OWASP Guidelines und Cheat Sheets	123
3.2.7 OWASP ESAPI	124
3.2.8 ÖNORM A7700	125
3.2.9 TSS-WEB	125
3.3 Übergreifende Sicherheitsprinzipien	125
3.3.1 Kenne deine Gegner	126
3.3.2 Berücksichtige Sicherheit im Entwurf („Secure by Design“)	128
3.3.3 Verwende einen offenen Entwurf	128
3.3.4 Verwende ein positives Sicherheitsmodell	129
3.3.5 Behebe die Ursachen (Ursachenbehebungsprinzip)	130
3.3.6 Minimiere die Angriffsfläche (Minimalprinzip)	131
3.3.7 Vermeide Risiken (Vermeidungsprinzip)	133
3.3.8 Verwende Indirektionen (Indirektionsprinzip)	133
3.3.9 Minimiere Privilegien („Least Privilege“)	134
3.3.10 Implementiere Sicherheit mehrschichtig („Defense in Depth“)	135
3.3.11 Gewährleiste einen sicheren Zustand („Fail Safe“)	137
3.3.12 Verwende sichere Standardeinstellungen („Secure Defaults“)	138
3.3.13 Vertraue niemandem (Misstrauensprinzip)	139
3.3.14 Gestalte Sicherheit konsistent	140
3.3.15 Vermeide Komplexität („Keep it Simple“)	141
3.3.16 Verwende ausgereifte Sicherheit	141
3.3.17 Gestalte Sicherheit anpassbar	142
3.3.18 Berücksichtige das Maximumprinzip	143
3.3.19 Antizipiere technologische Limitationen	143
3.3.20 Nutze Technologiebrüche	144
3.3.21 Stelle Testbarkeit sicher (Testbarkeitsprinzip)	144
3.3.22 Gestalte Sicherheit benutzerfreundlich	145
3.3.23 Trainiere deine Anwender	146
3.4 Anti-Patterns	147

3.5	Kryptographie	148
3.5.1	Zentrale kryptographische Verfahren	148
3.5.2	SSL- bzw. X.509-Zertifikate	148
3.5.3	HTTPS	151
3.5.4	Zusammenfassung	154
3.6	Datenvalidierung	155
3.6.1	Das Prinzip Datenvalidierung	156
3.6.2	Eingabevalidierung	158
3.6.3	Ausgabevalidierung	164
3.6.4	Schutz von Anwendungsparametern	171
3.6.5	Sonderfall Dateiuploads	172
3.6.6	Sonderfall URLs	174
3.6.7	Sonderfall Kommandos	175
3.6.8	Sonderfall XML	175
3.6.9	Sonderfall HTML-Markup	176
3.6.10	Überblick und Empfehlungen	178
3.7	Identifikation & Registrierung	179
3.7.1	Benutzerkennungen	179
3.7.2	Besucher-Tracking	180
3.7.3	IP-Adressen	181
3.7.4	Benutzerregistrierung durch technische Identifikation	182
3.7.5	Benutzerregistrierung durch persönliche Identifikation	183
3.7.6	Vorregistrierung	184
3.7.7	Gewinnspiele und Abstimmungen	184
3.7.8	Überblick und Empfehlungen	184
3.8	Authentifizierungsverfahren	186
3.8.1	IP-Adressen	187
3.8.2	HTTP Basic und HTTP Digest	188
3.8.3	Form-based Authentication	189
3.8.4	SSL-Client-Zertifikate (X.509-Zertifikate)	190
3.8.5	One Time Tokens (OTTs)	191
3.8.6	Kerberos (Windows-Authentifizierung)	192
3.8.7	SAML	193
3.8.8	OpenID	195
3.8.9	Mehrstufige Authentifizierung	196
3.8.10	Inter-Komponenten-Authentifizierung	198
3.8.11	Überblick und Empfehlungen	200
3.9	Benutzerpasswörter und Anmeldedialog	202
3.9.1	Passwortstärke	203
3.9.2	Generierte Passwörter	205
3.9.3	Passwort-Stärke-Funktionen	205
3.9.4	Neusetzen von Passwörtern (Passwort Reset)	207

3.9.5	Speicherung von Passwörtern (Passwort Hashing)	208
3.9.6	Überblick und Empfehlungen	210
3.10	Absicherung des Session Managements	212
3.10.1	Zufälligkeit der Session-ID	213
3.10.2	Härtung des Session Managements	213
3.10.3	Persistente Sessions	215
3.10.4	Session Binding (Browser Fingerprints)	216
3.10.5	Session-State-Kontrolle (CSRF- und Replay-Schutz)	216
3.10.6	Session Timeout	218
3.10.7	Mehrfachanmeldung (Concurrent Sessions)	220
3.10.8	Der Session-ID-Lifecycle	220
3.10.9	Shared Sessions	220
3.10.10	Überblick und Empfehlungen	221
3.11	Anti-Automatisierung	222
3.11.1	Limits	222
3.11.2	Verzögerungen	223
3.11.3	CAPTCHAS	223
3.11.4	Mehrstufigkeit	226
3.11.5	Weitere Verfahren	227
3.11.6	Überblick und Empfehlungen	228
3.12	Zugriffsschutz	230
3.12.1	Zugriffsmodelle	230
3.12.2	Mehrschichtige Zugriffskontrolle (expliziter Schutz)	231
3.12.3	Mehrschichtige Separierung (impliziter Schutz)	234
3.12.4	Rollen und Berechtigungen	235
3.12.5	Cross-Origin-Zugriffe	238
3.12.6	Access Tokens	243
3.12.7	OAuth	246
3.12.8	Geräte- bzw. Browser-Autorisierung	251
3.12.9	Überblick und Empfehlungen	252
3.13	Ereignisbehandlung	253
3.13.1	Fehlerbehandlung	254
3.13.2	Angriffserkennung und -behandlung (IDS/IPS)	255
3.13.3	User Alerting	257
3.13.4	Überblick und Empfehlungen	257
3.14	Datensicherheit und Datenschutz	258
3.14.1	Allgemeine Empfehlungen zum Schutz personenbezogener Daten	258
3.14.2	Datenverschlüsselung	260
3.14.3	Schutz der Integrität von Daten	264
3.14.4	Datenbehandlungsvorgaben	265
3.14.5	Datenschutzprofile	267

3.14.6	Abwicklung von Bezahlprozessen	268
3.14.7	Überblick und Empfehlungen	269
3.15	Clientseitige Sicherheitsaspekte	270
3.15.1	Gestaltung der Oberfläche	270
3.15.2	HTTP Strict Transport Security (HSTS)	272
3.15.3	Frame Busting	273
3.15.4	Schutz vor eingebettetem Schadcode (Werbebanner etc.)	275
3.15.5	Browserseitige XSS-Filter	276
3.15.6	Content Security Policy (CSP)	277
3.15.7	Dateidownloads	280
3.15.8	Browser-Plugins: Adobe Flash, Silverlight, ActiveX und Java-Applets	281
3.15.9	Visualisierung von Browsersicherheit	282
3.15.10	Überblick und Empfehlungen	282
3.16	Sicherheit von webbasierten Diensten	284
3.16.1	SOAP	285
3.16.2	XML-RPC	287
3.16.3	REST	287
3.16.4	JSON (Ajax)	288
3.16.5	WebSockets	289
3.16.6	RTMP	291
3.16.7	Überblick und Empfehlungen	292
3.17	Absicherung der Plattform	293
3.17.1	Generelle Architekturempfehlung der Infrastruktur	293
3.17.2	Härtung des SSL/TLS-Stacks	294
3.17.3	Härtung des Webservers	296
3.17.4	Laufzeitumgebung und Codeprivilegien	296
3.17.5	Webplattformen (WCMS-Systeme, Foren etc.)	302
3.17.6	Separierung (Abschottung)	304
3.17.7	Webanwendungsfirewalls (WAFs)	305
3.17.8	Cloud Computing	309
3.17.9	Überblick und Empfehlungen	311
3.18	Zusammenfassung	312
4	Sicherheitsuntersuchungen von Webanwendungen	315
4.1	Begriffe und Konzepte	315
4.1.1	Sicherheitsreview vs. Sicherheitstest	315
4.1.2	Risiko-basiertes Testen	316
4.1.3	Software Assurance (SwA)	316
4.1.4	Assurancegrad	316
4.1.5	Timeboxing	318
4.1.6	Low Hanging Fruits	319

4.2	Relevante Standards und Projekte	319
4.2.1	OSSTMM	319
4.2.2	OWASP ASVS Standard	319
4.2.3	OWASP Testing Guide	320
4.3	Lifecycle Security Testing	321
4.4	Wirksamkeit von Tools	323
4.5	Bewertungsverfahren	324
4.5.1	Risiken	324
4.5.2	CVSS	331
4.5.3	CWSS	332
4.5.4	DREAD	333
4.5.5	Bug Bars	335
4.5.6	Eignung der einzelnen Verfahren	336
4.6	Dynamische Testverfahren (Anwendungstests)	337
4.6.1	Verifikation von Sicherheitsvorgaben	338
4.6.2	Pentest (Penetrationstest)	340
4.6.3	Web Security Scanner (DAST)	346
4.6.4	Fault Injection (Fuzzing)	350
4.6.5	Security Integration Tests	352
4.6.6	Deployment Reviews (Konfigurationsanalysen)	353
4.7	Statische Testverfahren (Security Codeanalysen)	354
4.7.1	Security Codescanner (SAST)	354
4.7.2	Code Firewalls	361
4.7.3	Security Unit Tests	361
4.7.4	Security Code Review	363
4.8	Validierung von Sicherheitsanforderungen	365
4.9	Architekturelle Sicherheitsanalysen	367
4.9.1	Erstellen einer Übersicht der Sicherheitsarchitektur	368
4.9.2	Analyse der architekturellen Vertrauensbeziehungen	369
4.9.3	Weitere mögliche Analyseinhalte	370
4.10	Bedrohungs- und Risikoanalysen (Risk & Threat Assessments)	374
4.10.1	Ermittlung von Bedrohungen, Gefährdungen und Risiken	374
4.10.2	Existierende Vorgehensweisen	376
4.10.3	Ein generisches Vorgehensmodell	377
4.10.4	Verfahren zur Bedrohungsidentifikation	380
4.10.5	Mapping von Gegenmaßnahmen	383
4.10.6	Bedrohungskataloge (Threat Intelligence)	385
4.10.7	Übergang zur Risikoanalyse	387
4.10.8	Tools	388
4.10.9	Weitere Aspekte	388
4.11	Zusammenfassung	389

5 Organisatorische (Web-)Anwendungssicherheit	393
5.1 Begriffe und Konzepte	395
5.1.1 Application Security Management	395
5.1.2 Secure Software Development Lifecycle (SSDLC)	396
5.1.3 Reifegrade und Reifegradmodelle	397
5.2 Relevante Standards und Projekte	399
5.2.1 BSI Leitfäden zur Entwicklung sicherer Webanwendungen	399
5.2.2 OpenSAMM	399
5.2.3 BSIMM	400
5.2.4 ISO/IEC 27034	402
5.2.5 TSS-WEB	402
5.3 Maßnahmen zum Wissensaufbau und Wissenstransfer	402
5.3.1 Awareness und Management Attention schaffen	403
5.3.2 Schulungen durchführen	403
5.3.3 Multiplikatoren coachen	405
5.3.4 Lessons Learned und projektbezogene Workshops durchführen	405
5.3.5 Security Knowledge Base aufbauen	405
5.3.6 Mitarbeiter zertifizieren	406
5.4 Maßnahmen für das IT-Sicherheitsmanagement	406
5.4.1 Application Security Management Systems (ASMS) aufbauen	407
5.4.2 Richtlinien zur Anwendungssicherheit erstellen und pflegen	408
5.4.3 Vorgaben und Sign-Offs für Softwarelieferanten etablieren	409
5.4.4 Security Gates in Entwicklungs- und Beschaffungsprozesse integrieren	412
5.4.5 Risikomanagement für Anwendungssicherheit etablieren	414
5.4.6 Geschäftskritikalität und Schutzbedarf für Anwendungen spezifizieren	414
5.4.7 Zuständigkeiten für Anwendungssicherheit etablieren	415
5.4.8 Application Portfolio Management (APM) aufbauen	417
5.4.9 Project Portfolio Management (PPM) erweitern	418
5.4.10 Periodische Sicherheitsprüfungen aller Anwendungen durchführen	418
5.4.11 Application Incident und Problem Management einführen („Security Response“)	419
5.4.12 Key-Performance-Indikatoren (KPIs) definieren und reporten	420
5.5 Maßnahmen für das Projektmanagement	421
5.5.1 Zuständigkeiten und Sichtbarkeit für Sicherheit schaffen	422
5.5.2 Vertrauenswürdigkeit von Entwicklern sicherstellen	422
5.5.3 Fachliche Sicherheitsanforderungen und Risiken ermitteln	423
5.5.4 Sicherheitsumsetzungspläne erstellen und abstimmen	424
5.5.5 Sicherheitskonzepte erstellen und fortschreiben	424
5.5.6 Sicherheitskomponenten entkoppeln	425

5.5.7	Sicherheit im Change Management adressieren	426
5.5.8	Sicherheit in agile Vorgehensmodelle integrieren	427
5.5.9	Lessons Learned am Ende von Projekten durchführen	430
5.6	Maßnahmen für die Architektur	430
5.6.1	Architekturelle Vorgaben etablieren und kontrollieren	430
5.6.2	Sicherheitsdienste zentral bereitstellen	431
5.6.3	Technologiestacks und Blueprints vorgeben („Secure Foundation“)	432
5.6.4	Vorgaben und Sign-Offs für neue Technologien etablieren	433
5.7	Maßnahmen für die Entwicklung	436
5.7.1	Anreize schaffen	436
5.7.2	Software Security Group (SSG) aufbauen	436
5.7.3	Secure Coding Guidelines erstellen und pflegen	437
5.7.4	Vertrauen in die Entwicklungsumgebungen gewährleisten („Trusted Ecosystem“)	438
5.7.5	Sicherheitsfunktionen über Security-APIs zentral bereitstellen	439
5.7.6	Sicherheit bei Verwaltung von Sourcecode und Abhängigkeiten gewährleisten	439
5.7.7	Security Codeanalysen mit jedem Build durchführen	441
5.7.8	Sicherheit im Deployment-Prozess berücksichtigen	442
5.7.9	Kollaborative Sicherheitsreviews durchführen	442
5.7.10	Hacker-Techniken und -Tools einsetzen	442
5.7.11	Sicherheitsaspekte dokumentieren	443
5.8	Maßnahmen für die Qualitätssicherung	443
5.8.1	Produktiv- von Nicht-Produktiv-Systemen abschotten	444
5.8.2	Sicherheitskennzahlen spezifizieren und überwachen	444
5.8.3	Security Testing Factory aufbauen	446
5.8.4	Infrastruktur für Security-Scans bereitstellen	448
5.8.5	Sicherheitstests in verwendete Toolsuiten integrieren	449
5.8.6	Testbarkeit gewährleisten	450
5.8.7	Sicherheitstestplanung einfordern	450
5.9	Maßnahmen für den Betrieb	451
5.9.1	Plattform absichern	451
5.9.2	Laufende Sicherheitsscans der Plattform durchführen	451
5.9.3	Patch Management einbeziehen für 3rd-Party-Komponenten etablieren	452
5.9.4	Virtuelles Patch Management und Application Incident Response gewährleisten	453
5.9.5	Application IDS betreiben (Security Logging & Monitoring)	454
5.9.6	Zertifikatsmanagement betreiben	456
5.10	Maßnahmen für Softwarehersteller	456

5.11 Durchführen eines Programms zur Steigerung der Anwendungssicherheit (ASP)	457
5.11.1 Erfolgsfaktoren	457
5.11.2 Formelles Vorgehensmodell	458
5.11.3 Empfehlung für die praktische Durchführung eines ASPs	462
5.12 Zusammenfassung	464
6 Schlussbemerkungen	469
Erratum	E1
Anhang: Mapping von Maßnahmen zur OWASP Top 10	471
Glossar	473
Literatur	491
Sachverzeichnis	499

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.

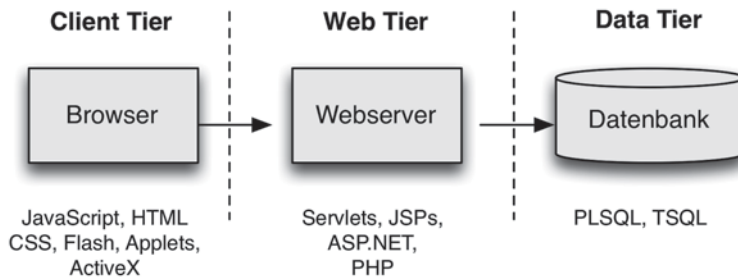


Abb. 1.1 3-Tier-Architektur einer Webanwendung

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 Applikationsservern¹ 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.

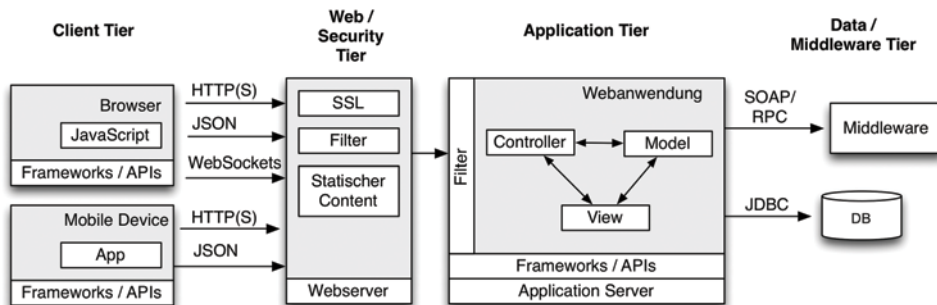


Abb. 1.2 N-Tier-Architektur einer Enterprise-Webanwendung

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,

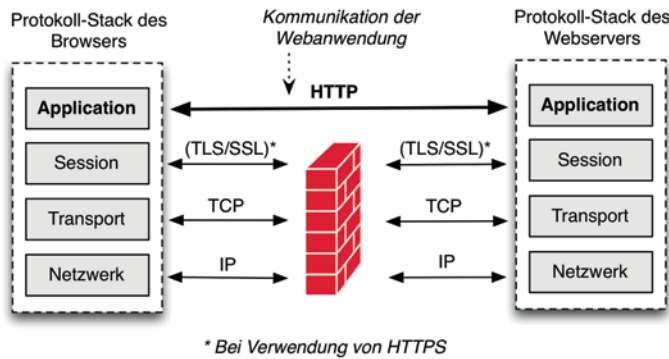


Abb. 1.3 Netzwerk- vs. Anwendungslayer

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 Übertragungsprotokoll². 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
```

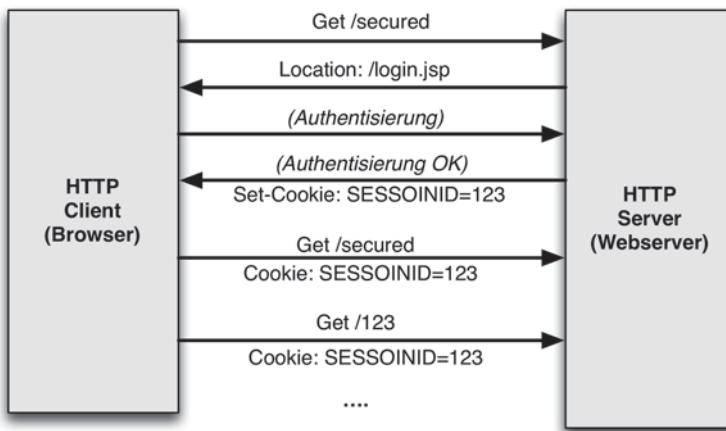


Abb. 1.4 Funktionsweise eines Cookie-basierten Session Managements

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 Sicherheitsaspekte⁵, 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.

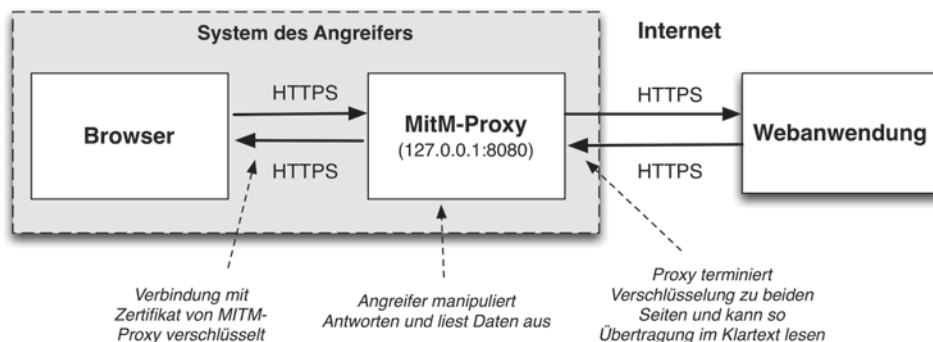


Abb. 1.5 Funktionsweise eines MitM-Proxys

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.

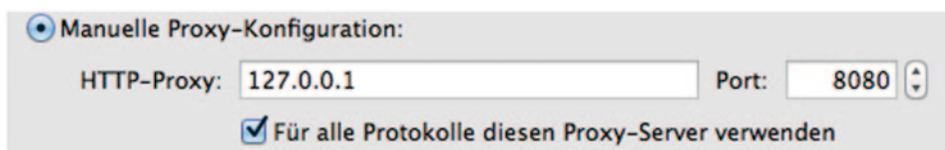


Abb. 1.6 Einbinden eines MitM-Proxys im Browser

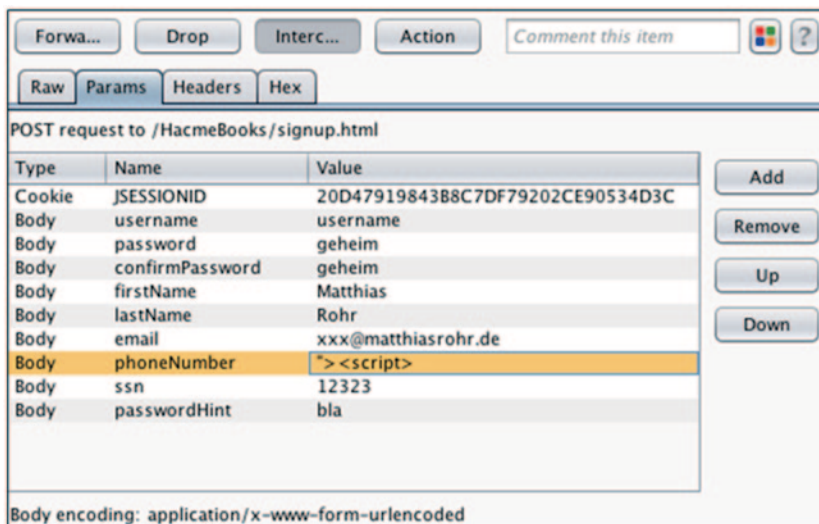


Abb. 1.7 Burp Proxy

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