Diese Leseprobe haben Sie beim edv-buchversand.de heruntergeladen.
Das Buch können Sie online in unserem Shop bestellen.

Hier zum Shop

Kapitel 1
Über dieses Buch

Webanwendungen enthalten wie jede andere Software auch mehr oder weniger viele Fehler. Manche davon lassen sich für Angriffe ausnutzen, und die nennt man dann Schwachstellen. Die entscheidenden Fragen sind: Welche Schwachstellen gibt es? Wo befinden sie sich? Und wie kann man entweder die Schwachstelle beseitigen oder zumindest die Angriffe darauf abwehren?

1.1 Was dieses Buch ist

In diesem Buch zeige ich Ihnen, wie Sie Schwachstellen in Ihrer Webanwendung finden.

Zuvor muss ich Ihnen natürlich erklären, welche Schwachstellen es überhaupt gibt, schließlich ist es ziemlich schwierig, etwas zu finden, worüber man gar nichts weiß.

Natürlich gibt es auch Hinweise darauf, wie Sie die jeweiligen Schwachstellen beseitigen oder verhindern können oder, wenn das nicht möglich ist, Angriffe abwehren oder zumindest erschweren.

Dazu gibt es eine kleine Demo-Webanwendung voller Schwachstellen, an der Sie Ihre neu gewonnenen Fähigkeiten testen können.

1.1.1 Was dieses Buch nicht ist

Dieses Buch ist keine Einführung in die Programmierung oder auch nur die sichere Programmierung. Sie müssen auch nicht unbedingt programmieren können, um die Schwachstellen zu verstehen und anschließend danach zu suchen. Programmierkenntnisse schaden natürlich nicht, aber auch ohne sollten Sie die Probleme erkennen können.

Dieses Buch ist auch keine Einführung in die Konfiguration oder Administration von Webanwendungen. Hier gilt ebenso: Sie müssen kein Admin sein, um die Schwachstellen zu verstehen und zu finden. Auch hier schaden Grundkenntnisse nicht, aber auch ohne sollten Sie keine Schwierigkeiten haben, dem Text zu folgen.

1.1.2 Wie ist das Buch aufgebaut?

Oder genauer: Warum ist es so aufgebaut, wie es aufgebaut ist?

Ich werde der Reihe nach typische Schwachstellen in Webanwendungen vorstellen und beschreiben, wie sie entstehen und wie sie von Angreifern ausgenutzt werden können (und teilweise anhand realer Angriffe auch mit welchen Folgen). Danach zeige ich Ihnen, wie Sie sie allgemein finden und welche Möglichkeiten Sie haben, die Schwachstellen zu beseitigen und Angriffe darauf abzuwehren oder zumindest so schwer wie möglich zu machen.

Danach können Sie die Suche und das Testen dieser Schwachstellen an der Demo-Anwendung ausprobieren.

Die Gliederung orientiert sich am Vorgehen eines Penetration Testers oder Angreifers, denn das Vorgehen ist immer gleich – egal, ob jemand wie ich nach Schwachstellen sucht, damit sie hinterher beseitigen werden können, oder ob ein Cyberkrimineller nach einer Möglichkeit sucht, eine Webanwendung oder einen Server unter seine Kontrolle zu bringen. Außerdem ist es generell keine schlechte Idee, sich in die Lage des Angreifers zu versetzen, wenn man nach Möglichkeiten sucht, den Angriff zu verhindern. Details dazu gibt es in Abschnitt 1.2.3.

1.1.3 Danksagungen

Ich bedanke mich bei Almut Poll und Christoph Meister vom Rheinwerk Verlag für die gute Zusammenarbeit und ihre Geduld, als Murphy mehrmals erfolgreich die Fertigstellung verzögerte. Aber wie Sie sehen, lassen wir uns auch von Murpyhs Law nicht aufhalten.

Außerdem gilt mein Dank natürlich auch allen anderen an der Entstehung Beteiligten, vor allem dem Gutachter für seine hilfreichen Kommentare und der Korrektorin für das Ausbügeln meiner Fehler (Alle, die jetzt noch drin sind, gehen auf meine Kappe!).

Ein besonderer Dank gilt meinem kleinen »Helferlein« für seine hilfreiche Unterstützung.

1.2 Welche Schwachstellen gibt es?

Das Open Web Application Security Project (OWASP) veröffentlicht alle paar Jahre eine Liste mit den Top 10 der größten Bedrohungen für Webanwendungen [OWASP_Top10][1]. Aktuell ist derzeit die Version 2017, die im Herbst 2017 veröffentlicht wurde [OWASP_Top10-2017]. Aber wie es bei Top 10 üblich ist, gibt es noch viel mehr Bedrohungen, Schwachstellen,

1 Ich verwende »sprechende« Kürzel für die Linkverweise. Teilweise so wie hier mit einem das Ziel beschreibenden Begriff, teilweise nutze ich die Notation, die ich noch aus alten Uni-Zeiten gewohnt bin: [<Erste 4 Zeichen des Nachnamens>_<Veröffentlichungsjahr>] oder bei mehreren Autoren die Anfangsbuchstaben der Autoren. Wichtige Links finden Sie auch im Fließtext, die vollständige Auflistung immer am Ende des Kapitels.

Angriffe ... Und ob die Top 10 wirklich so eine große Bedrohung sind, ist auch immer relativ. Auf Platz 4 stehen die XML External Entities (XXE). Für Webanwendungen, die XML gar nicht nutzen, ist das völlig harmlos. Was nicht da ist, kann auch nicht angegriffen werden.

Das ist aber auch nicht weiter schlimm, die OWASP Top 10 sollen ja gar nicht dazu dienen, z. B. Schwachstellentests zu planen oder die Sicherheit einer Webanwendung zu bewerten, sondern das Bewusstsein für diese Schwachstellen zu schärfen. Woraufhin diese hoffentlich seltener werden und dann irgendwann aus den Top 10 herausfallen.

Auch in dieser Version gab es einige Änderungen zur Vorgängerversion von 2013. So ist z. B. das Cross-Site-Scripting von Platz 3 auf Platz 7 gefallen, dafür ist der Punkt »Sensitive Data Exposure«/»Preisgabe sensibler Daten« von Platz 6 auf Platz 3 aufgestiegen. Die Cross-Site-Request-Forgery, die 2013 noch auf Platz 8 stand, ist inzwischen aus den Top 10 heraus und ungefähr auf Platz 13 gelandet [OWASP_Top10-2017-RC2]. Wenn Sie jetzt nur Bahnhof verstanden haben: Keine Angst, die ganzen Schwachstellen werde ich im Laufe des Buchs noch erklären.

Aber weil ich gerade von der Bewertung der Sicherheit von Webanwendungen geschrieben habe: Wieso muss ich da bloß immer an die Quartette aus meiner Kindheit denken? »Ich habe drei XSS und eine Sensitive Data Exposure« – »Und ich habe drei SQL-Injections und zwei OS-Command-Injections« – »OK, Du hast gewonnen, hier ist meine Karte. Du bist dran!«

Vermutlich, weil ich das für Spielerei halte. Jede Software und damit auch jede Webanwendung sollte möglichst gar keine Schwachstellen enthalten, egal, wie leicht oder schwer sie auszunutzen und wie gefährlich die Folgen eines Angriffs sind.

Natürlich ist es schlimm, wenn über z. B. eine SQL-Injection-Schwachstelle sämtliche Datenbanken ausgespäht werden. Aber was ist, wenn über eine persistente XSS-Schwachstelle Code zur Infektion der Rechner aller Besucher der Website eingeschleust wird – was ist dann schlimmer?

Im ersten Fall betrifft es einige Tausend registrierte Benutzer, deren Zugangsdaten und Kreditkartennummern von den Cyberkriminellen gehandelt werden. Im zweiten Szenario sind es einige zig Tausend arglose Besucher, deren Rechner nach dem Besuch mit einem Onlinebanking-Trojaner infiziert ist. Was ist schlimmer?

Oder vielleicht wurde die Webanwendung ja auch von einem Geheimdienst über einen Fehler in der Authentifizierung gezielt infiziert, um sie für den Angriff auf den Rechner eines Dissidenten zu präparieren? Sie wissen ja: Die Dissidenten und Terroristen der einen Seite sind für die andere unschuldig Verfolgte und Freiheitskämpfer. Wie soll man das bewerten?

Ich denke, wir sind uns hier einig: Jede einzelne Schwachstelle in einer Webanwendung ist eine zu viel. Und Ranglisten haben immer etwas Subjektives, über das man sich streiten kann.

1.2.1 Die OWASP Top 10

In den OWASP Top 10 vom 2017 [OWASP_Top10-2017] sind folgende Bedrohungen enthalten:

- A1:2017 – **Injection**

  Enthält Schwachstellen wie die SQL-Injection (siehe Kapitel 6), OS-Command- und SMTP-Injection (beide in Kapitel 7) und andere Schwachstellen, bei denen der Angreifer eigene Befehle in Interpreter und Ähnliches einschleusen kann

- A2:2017 – **Broken Authentication** (gebrochene Authentifizierung)

  Wer Benutzer hat, muss sie identifizieren können. Das nennt man *Authentifizierung*, und jeder Fehler darin erlaubt es einem Angreifer, sich als anderer Benutzer auszugeben und/oder unbefugt auf die Webanwendung zuzugreifen (siehe Kapitel 4, »Angriffe auf die Authentifizierung«).

- A3:2017 – **Sensitive Data Exposure** (Preisgabe sensibler Daten)

  Das ist ein weites Feld. Dazu gehören unverschlüsselt übertragene Zugangsdaten, die dann unterwegs belauscht werden können, oder offen zugängliche Dateien mit vertraulichen Daten (siehe z. B. Kapitel 6, »SQL-Injection«).

- A4:2017 – **XML External Entities (XXE)**

  Wird XML verwendet, kann es über XML External Entities manipuliert werden. XML und damit XXE kommen hier im Buch nicht vor.

- A5:2017 – **Broken Access Control** (gebrochene Zugriffskontrolle)

  Die Authentifizierung stellt sicher, dass Sie wissen, wer der Benutzer ist. Genauso wichtig ist aber die Zugriffskontrolle oder Autorisierung: Darf der Benutzer das, was er gerade machen will, überhaupt? Hier im Buch kommt das z. B. in Kapitel 3, »Zustandsbasierte Angriffe«, vor.

- A6:2017 – **Security Misconfiguration** (sicherheitsrelevante Fehlkonfiguration)

  Alles, was sich konfigurieren lässt, lässt sich auch unsicher konfigurieren, z. B. wenn für die TLS-Verschlüsselung unsichere Kryptoverfahren verwendet werden oder wenn ein eigentlich nötiger Passwortschutz vergessen wird, so dass Teile der Webanwendung frei zugänglich sind, statt wie geplant nur bestimmten Benutzern offenzustehen. Das kommt im Buch an verschiedenen Stellen vor.

- A7:2017 – **Cross-Site-Scripting (XSS)**

  Das Einschleusen von bösartigem JavaScript-Code in die Webanwendung erlaubt alle möglichen Angriffe, beispielsweise kann darüber der Benutzer ausgespäht oder sein Rechner mit Schadsoftware infiziert werden.

- A8:2017 – **Insecure Deserialization** (unsichere Deserialisierung)

  Werden Daten serialisiert gespeichert (vereinfacht werden dabei komplizierte Strukturen einfach hintereinander weg in eine Variable geschrieben), müssen sie später wieder deserialisiert, d. h. in die Ausgangsdaten zurückverwandelt werden. Ein Angreifer kann eine Schwachstelle dabei ausnutzen, um Daten zu manipulieren. Dies kommt im Buch nicht vor.

- A9:2017 – **Using Components with known Vulnerabilities** (Nutzung von Komponenten mit bekannten Schwachstellen)

  Wenn Sie die Demo-Anwendung als Blog in Ihre Webanwendung integrieren, haben Sie dieses Problem. Spaß beiseite: Hier geht es wirklich darum, dass z. B. Bibliotheken mit bekannten Schwachstellen verwendet werden. Das Problem sind dabei weniger die direkt genutzten Komponenten, die kennen Sie hoffentlich und können sie aktuell halten. Aber wie sieht es mit den Dritthersteller-Komponenten aus, die diese von Ihnen genutzten Komponenten verwenden? Wer sorgt dafür, dass diese immer in einer sicheren Version verwendet werden? Sie können das kaum, dazu müssten Sie nämlich wissen, welche Komponenten das sind, und wenn es ein Update gibt, prüfen, ob Sie das einfach so installieren können oder ob danach irgendetwas mit der Komponente nicht mehr funktioniert. Das wird nie was. Im Buch kommt dies Thema aber nicht vor.

- A10:2017 – **Insufficient Logging & Monitoring** (unzureichende Protokollierung und Überwachung)

  Das ist wie die XXE auf Platz 4 ein Neuzugang in den Top 10 2017. Da es sich weder um eine Schwachstelle in der Webanwendung noch um einen Angriff darauf handelt, passt es eigentlich nicht richtig in die Liste. Dabei ist es doch sehr wichtig, denn wenn Sie nicht merken, dass jemand versucht, Ihre Webanwendung anzugreifen, können Sie den Angriff auch nicht abwehren. Dieses Thema werde ich in Abschnitt 1.6 kurz behandeln.

1.2.2 Weitere Schwachstellen

Außerdem stelle ich im Buch Schwachstellen vor, die nicht in den Top 10 enthalten oder gerade hinausgerutscht sind, z. B. Clickjacking, das es nie in die Top 10 geschafft hat und unter dessen Variante »Likejacking« – Angriffe auf den »Like«-Button – Facebook sehr gelitten hat. Ich würde wetten, dass Mark Zuckerberg Clickjacking ziemlich weit oben in den Top 10 sehen würde. Auch die schon erwähnte CSRF, die 2017 aus den Top 10 ausschied, kommt im Buch vor.

1.2.3 Gliederung des Buchs

Die Gliederung richtet sich nach einem möglichen Vorgehen bei der Suche nach Schwachstellen: Zunächst erkunden Sie die Webanwendung, dann suchen Sie entweder systematisch schrittweise nach den verschiedenen Schwachstellen (wenn Sie alle finden möchten) oder gezielt nach bestimmten Schwachstellen (wenn Sie beispielsweise nur Angriffe auf die Datenbank durch SQL-Injection untersuchen möchten). So eine Suche nach Schwachstellen wird auch als *Penetrationstest* bezeichnet.

Auch ein Angriff beginnt meist mit der Erkundung der Anwendung, danach sucht der Angreifer gezielt nach den für seinen Zweck benötigten Schwachstellen. Möchte er sich Zugriff auf die Anwendung verschaffen, wird er meist zunächst die Authentifizierung auf Schwach-

stellen abklopfen und erst, wenn er an dieser »Vordertür« nicht fündig wird, nach anderen Möglichkeiten zum Eindringen in den Webserver suchen. Möchte er die Webanwendung »nur« für eine Drive-by-Infektion präparieren, reicht ihm dafür auch eine persistente Cross-Site-Scripting-Schwachstelle aus. Was diese Fachbegriffe bedeuten, erfahren Sie noch in den entsprechenden Kapiteln. Hier reicht es, wenn Sie wissen, dass es sich dabei um Schwachstellen bzw. Angriffe handelt – und dass ein Angreifer nichts anderes macht als ein Penetration Tester. Nur eben deutlich zielgerichteter, da er ja nur eine einzige Schwachstelle braucht, über deren Ausnutzung er sein Ziel erreichen kann. Beim Penetrationstest dagegen sollen ja möglichst alle Schwachstellen gefunden werden.

Die Gliederung im Überblick

1. Schritt: Das Ziel erkunden.

   Was ist das für eine Webanwendung, wie ist sie aufgebaut, welche Parameter gibt es? Kurz: so viel Wissen wie möglich über das Angriffsziel sammeln. Darum geht es in Kapitel 2.

2. Schritt: Testen von zustandsbasierten Angriffen

   Oft gibt es Funktionen in Webanwendungen, die nur dann das gewünschte Ergebnis liefern, wenn einige Schritte in der richtigen Reihenfolge durchlaufen werden. Dazu gehört der Einkauf in einem Webshop: Artikel auswählen und die gewünschten Anzahl angeben, Lieferadresse und Kreditkartendaten eingeben, Bestellung abschicken, das sollte immer in dieser Reihenfolge ablaufen. Wenn ein Angreifer Punkte dieser Liste auslassen oder in einer anderen Reihenfolge durchlaufen kann, hat das meist unerwünschte Folgen, vor allem natürlich, wenn er die Eingabe der Kreditkartendaten überspringen kann, aber auch die Änderung der Menge nach erfolgter Bezahlung wäre nicht besonders schön. Es gibt noch weitere zustandsbasierte Angriffe, und um sie geht es in Kapitel 3.

3. Schritt: Angriffe auf die Authentifizierung

   Viele Webanwendungen erfordern eine Authentifizierung, also die Anmeldung des Benutzers, damit sie wirklich genutzt werden können. Kann ein Angreifer die Authentifizierung manipulieren, kann er sich als ein anderer Benutzer ausgeben und unbefugt Zugriff auf die Anwendung erhalten. Diese Angriffe behandele ich in Kapitel 4.

4. Schritt: Cross-Site-Scripting (XSS)

   Als XSS wird das Einschleusen von JavaScript- und HTML-Code in die Webanwendung bezeichnet. Wenn Sie sich überlegen, was man heutzutage mit den Webclients alles machen kann, können Sie sich sicher vorstellen, dass es gar nicht gut ist, wenn sich ein Angreifer daran zu schaffen macht. Zum Glück sind XSS-Schwachstellen inzwischen seltener geworden, und damit sie noch seltener werden, haben sie ein eigenes Kapitel bekommen: Kapitel 5.

5. Schritt: SQL-Injection

   Die meisten Webanwendungen speichern Daten in Datenbanken, und auf diese wird im Allgemeinen mit der Abfragesprache SQL zugegriffen. Kann ein Angreifer in die Abfragen

der Webanwendung eigene Befehle einschleusen, kann er die Datenbank ausspähen oder manipulieren. Damit Ihrer Webanwendung das nicht passiert, erfahren Sie in Kapitel 6 alles Nötige über SQL-Injection und das Finden und Verhindern dieser Schwachstellen.

6. Schritt: weitere Injection-Schwachstellen

   Nicht nur in SQL-Abfragen, auch in vielen anderen Fällen kann ein Angreifer irgendwo eigene Befehle einschleusen, z. B. in die Aufrufe von Shellbefehlen, die Kommunikation mit dem Mailserver, LDAP-Abfragen ... – diese Angriffe stelle ich in Kapitel 7 vor.

7. Schritt: Dateioperationen

   Wenn die Webanwendung auf Dateien zugreift und ein Angreifer diese Zugriffe manipuliert, kann alles Mögliche passieren, und das meiste davon ist unschön, beispielsweise wenn ein Webshop eigentlich eine Datei mit den allgemeinen Geschäftsbedingungen in der Sprache des Benutzers laden möchte, der Angreifer den Zugriff aber so manipuliert, dass stattdessen eine Datei mit Zugangsdaten angezeigt wird. Angriffe auf Dateioperationen sind in Kapitel 8 an der Reihe.

8. Schritt: Pufferüberläufe, Format-Strings und Integer-Fehler

   Diese Schwachstellen kommen in Webanwendungen selbst eigentlich nicht vor. Aber die Webanwendungen nutzen Laufzeitumgebung, Bibliotheken ..., die eventuell von solchen Schwachstellen betroffen sind. Und dann kann ein Angreifer sie ja vielleicht über Eingaben an die Webanwendung angreifen. Kapitel 9 gibt einen kleinen Überblick über diese Schwachstellen.

9. Schritt: Angriff auf die Architektur der Webanwendung

   Webanwendungen bestehen oft aus mehr als einer Komponente und mehr als einer Schicht. Wenn der Angreifer an einer Stelle nicht weiterkommt, dann vielleicht an einer anderen. Kapitel 10 gibt einen kurzen Überblick über die möglichen Probleme.

10. Schritt: Angriffe auf den Webserver

    Jede Webanwendung ist auf einen Webserver angewiesen. Falls er bekannte Schwachstellen enthält, ist es für den Angreifer mitunter leichter, den Webserver statt der Webanwendung anzugreifen: Wenn er den Serverprozess kontrolliert, kann er darüber auch die Webanwendung kontrollieren. Solche Angriffe auf Webserver erläutere ich in Kapitel 11.

1.3 Wie findet man Schwachstellen?

Die Antwort auf diese Frage ist eigentlich ganz einfach: Entweder prüfen Sie den Quellcode auf Programmierfehler, die zu einer Schwachstelle führen, d. h. im Wesentlichen fehlende oder unzureichende Prüfungen von Eingabewerten oder eigentlich nötige, aber nicht vorhandene Schutzmaßnahmen. Oder Sie geben für alle in der Webanwendung verwendeten Parameter mögliche Angriffsmuster ein und schauen, was passiert. Natürlich können Sie auch beide Ansätze kombinieren.

1.3.1 Manuelle oder automatische Suche?

Sie können nach den Schwachstellen manuell suchen (und darum geht es in diesem Buch) oder ein Tool mit der automatischen Suche beauftragen. Die einzigen wesentlichen Unterschiede zwischen einer manuellen Suche und der Arbeitsweise eines Schwachstellen-Scanners sind die menschliche Intuition und das Verständnis des Kontexts einer Webseite bzw. Funktion auf der einen Seite und die Unermüdlichkeit des Rechners auf der anderen.

Ein Schwachstellen-Scanner nutzt mehr oder weniger dieselben Arbeitsweisen wie ein Mensch, kann aber Schwachstellen nur aufgrund vorgegebener Regeln erkennen, während ein Mensch auch vollkommen neue Schwachstellen finden kann. Während ein Mensch Rücksicht auf den aktuellen Kontext nehmen kann und nur die Tests durchführt, die in der aktuellen Situation nötig sind, probiert der Rechner meist einfach alle Möglichkeiten durch. Dafür ermüdet der Rechner nicht so schnell wie ein Mensch, weshalb dieser sich mitunter auch vom Rechner helfen lassen wird, z. B. wenn es gilt, alle Seiten einer Webanwendung zu suchen und die Parameter zu sammeln. Das kann ein Webcrawler deutlich schneller als ein Mensch mit seinem Browser, und wenn keine weiteren Informationen aus den Seiten benötigt werden, ist der Crawler deutlich im Vorteil.

1.4 Einige Tools im Überblick

Für die Suche benötigen Sie einige Tools, die ich Ihnen hier kurz vorstellen werde. Ich behandle sie aber nur so weit, wie es nötig ist, damit Sie sie für die Suche und Tests nutzen können. Die Tools können natürlich alle noch viel mehr, aber das müssten Sie dann bitte bei Interesse in der Dokumentation, How-tos und FAQs nachlesen.

Die Tools können Sie einzeln auf Ihrem eigenen Rechner installieren, oder Sie verwenden Kali Linux, eine spezielle Linux-Live-Distribution für das Penetration Testing, die die Tools von Haus aus mitbringt. Das bleibt sich im Grunde gleich. Das Gleiche gilt für die Demo-Anwendung, auch sie können Sie auf einem eigenen Server installieren oder auf einer virtuellen Maschine. Ich selbst nutze eine Kombination von allem: Die Demo-Anwendung läuft in einer virtuellen Maschine, die Tools habe ich zum Teil auf dem Rechner installiert, zum Teil verwende ich Kali Linux in einer zweiten virtuellen Maschine.

1.4.1 Der OWASP Zed Attack Proxy (ZAP)

Das Tool, das Sie am häufigsten einsetzen werden, ist der *OWASP Zed Attack Proxy* (ZAP), das Sie unter *https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project* finden [OWASP_ZAP]. Deshalb beschreibe ich ihn hier etwas ausführlicher.

Der ZAP ist inzwischen viel mehr als ein einfacher Proxy, für die Arbeit mit der Demo-Anwendung brauchen Sie ihn aber nur dafür. Dass er auch als Crawler oder Schwachstellenscanner arbeiten kann und viele weitere Funktionen und Tools enthält, ist hier nicht relevant. Aber Sie könnten später beispielsweise Ihre manuellen Testergebnisse mit denen des ZAP vergleichen oder ihn für weitergehende Untersuchungen verwenden.

Wofür wird der Proxy benötigt?

Der Proxy wird vor allem für zwei Einsatzzwecke benötigt: Zum einen müssen Sie immer wieder die Header der HTTP-Requests oder -Responses betrachten oder auch manipulieren. Zum anderen müssen Sie für die meisten Tests die Parameter der Requests manipulieren. Das geht bei GET-Requests in der URL-Zeile des Browsers, bei POST-Requests aber nur im Proxy. Und auch an die im Header übertragenen Cookies kommen Sie nur über den Proxy.

Dieser hängt in der Verbindung zwischen Webbrowser und Webanwendung und kann die Requests und Responses generell oder auch gezielt aufhalten und anzeigen. Dann können Sie sie in Ruhe studieren und/oder manipulieren, bevor Sie sie an ihr Ziel weiterleiten lassen.

Den ZAP installieren und konfigurieren

Der Proxy muss auf dem Rechner laufen, auf dem auch Ihr Browser läuft. Wenn Sie den ZAP also nicht auf Ihrem System installieren möchten, sondern die in Kali Linux vorhandene Installation verwenden wollen, müssen Sie auch den Browser von Kali Linux verwenden.

Wenn Sie den ZAP auf Ihrem System installieren wollen, laden Sie die passende Version herunter [OWASP_ZAP]. ZAP ist so wie viele Security-Tools eine Java-Anwendung, und während die Mac-Version bereits eine passende Java-Version enthält, müssen Sie unter Windows und Linux gegebenenfalls Java Version 8 oder höher installieren. Wenn Java installiert ist, führen Sie den Installer aus.

Nachdem ZAP installiert ist, gibt es (abgesehen von Unterschieden durch verschiedene System- und Programmversionen) keinen Unterschied mehr zwischen einer Installation auf Ihrem Rechner und der Version von Kali Linux.

Nach dem ersten Start müssen Sie den Lizenzbedingungen zustimmen, danach werden Sie gefragt, ob Sie die neu gestartete Session speichern wollen (Do you want to persist the ZAP Session?). Das ist nicht nötig, also bestätigen Sie die Defaultauswahl No, ... mit Klick auf Start. Eine kurze Einführung in ZAP liefert der auf der Download-Seite verlinkte *OWASP ZAP 2.7 Getting Started Guide*. Sie brauchen nur die Proxyfunktion, und die ist schnell erklärt.

ZAP besitzt eine Schnellstart-Funktion, siehe ❶ in Abbildung 1.1. Dort können Sie einen Browser auswählen und starten, der dann sofort richtig konfiguriert ist, um den ZAP-Proxy zu nutzen.

Die Schnellstart-Funktion können Sie nutzen, ich selbst konfiguriere meinen Browser bei Bedarf selbst so, dass er den ZAP als Proxy nutzt. Im Firefox 62 müssen Sie dazu nur in den Einstellungen unter Allgemein ganz unten die Einstellungen für den Netzwerkproxy öffnen und dort die manuelle Proxykonfiguration auswählen. Der Proxy ist dann der localhost mit Port 8080, siehe Abbildung 1.2.

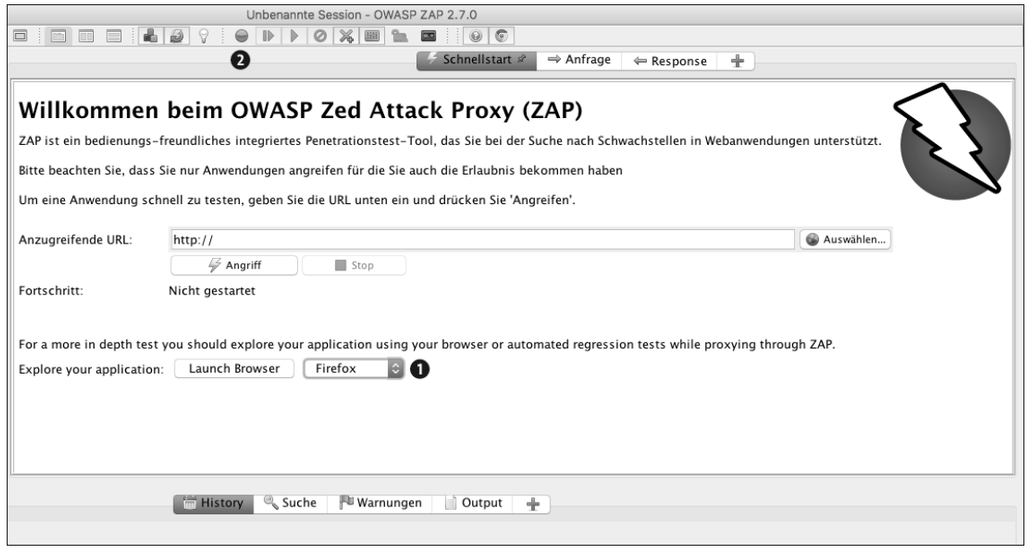

Abbildung 1.1 Die Schnellstart-Funktion und die Breakpoint-Auswahl des ZAP

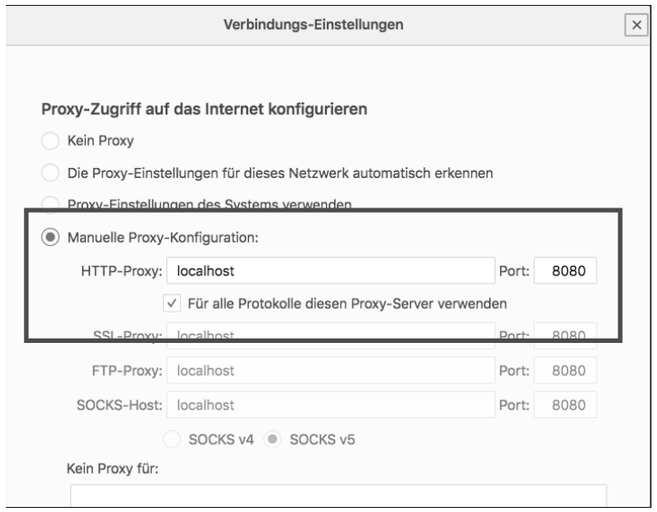

Abbildung 1.2 Die Proxyeinstellungen des Firefox 62

Im Firefox 52.5 ESR von Kali Linux erfolgt die Proxykonfiguration über die CONNECTION-Einstellungen im NETWORK-Tab unter ADVANCED, siehe Abbildung 1.3. Auch hier müssen Sie die manuelle Konfiguration auswählen, und auch hier ist der PROXY der LOCALHOST mit `Port` 8080, siehe Abbildung 1.4.

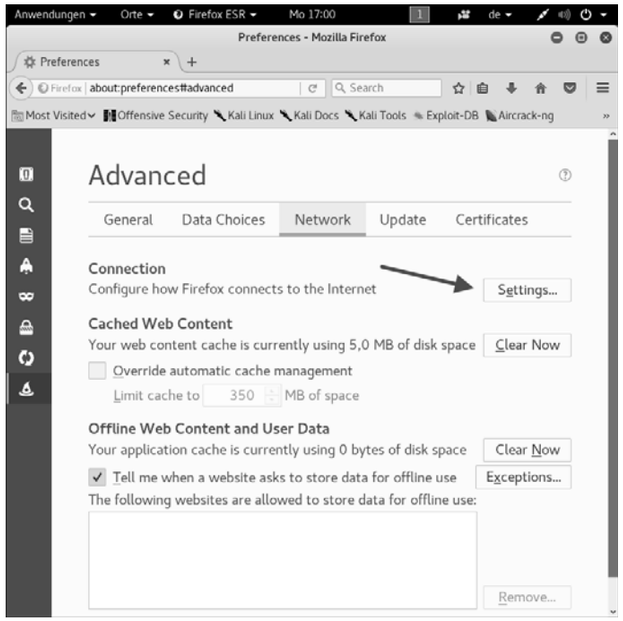

Abbildung 1.3 Firefox 52.5 ESR von Kali Linux: der Weg zur Proxykonfiguration

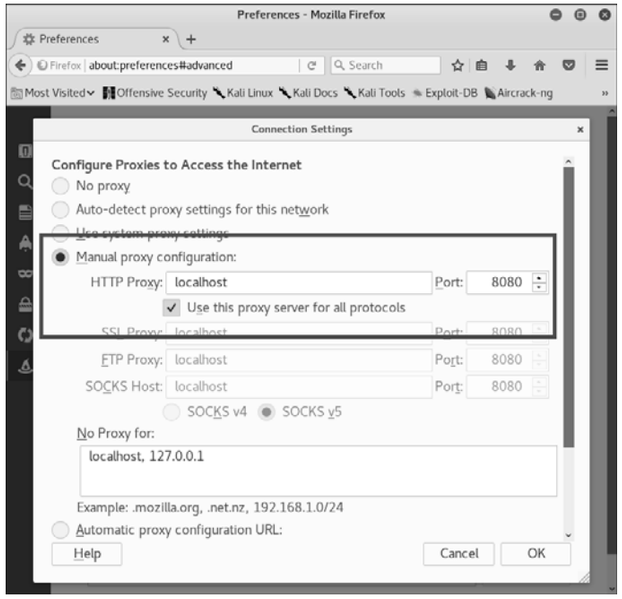

Abbildung 1.4 Die Proxyeinstellungen des Firefox 52.5 ESR von Kali Linux

Um auch HTTPS-Verbindungen nutzen zu können, müssten Sie jetzt noch das Root-Zertifikat des ZAP in Firefox installieren. Da die Demo-Anwendung aber kein HTTPS nutzt, können Sie darauf problemlos verzichten.

Sie sollten während der Tests bzw. während der Proxy läuft mit dem dafür konfigurierten Browser nicht zusätzlich im Internet surfen. Bei HTTPS-Links gibt es dann eine Warnung, bei gesetztem Breakpoint ist es unpraktisch, und außerdem »vermüllt« das die ZAP-Session, falls Sie sich später überlegen, diese doch zu speichern – der ZAP-Proxy protokolliert unten im Fenster alle Requests und Responses, so dass Sie darauf später zurückgreifen können.

Nachdem Sie den Browser konfiguriert haben, können Sie loslegen. Bis jetzt hat der Proxy alle Requests durchgelassen; wenn Sie ein paar Websites aufgerufen haben, ist das ZAP-Fenster in der unteren Hälfte schon gut gefüllt. Damit ist jetzt aber Schluss, denn ab sofort soll der Proxy alle Requests und Responses aufhalten. Dazu müssen Sie einen Breakpoint setzen. Das ist der zurzeit grüne Punkt über dem oberen rechten Feld, siehe ❷ in Abbildung 1.1. Klicken Sie ihn an, so dass er grün wird.

Nun können Sie die Demo-Anwendung im Browser aufrufen. Der Proxy hält den Request auf und drängelt sich in den Vordergrund.

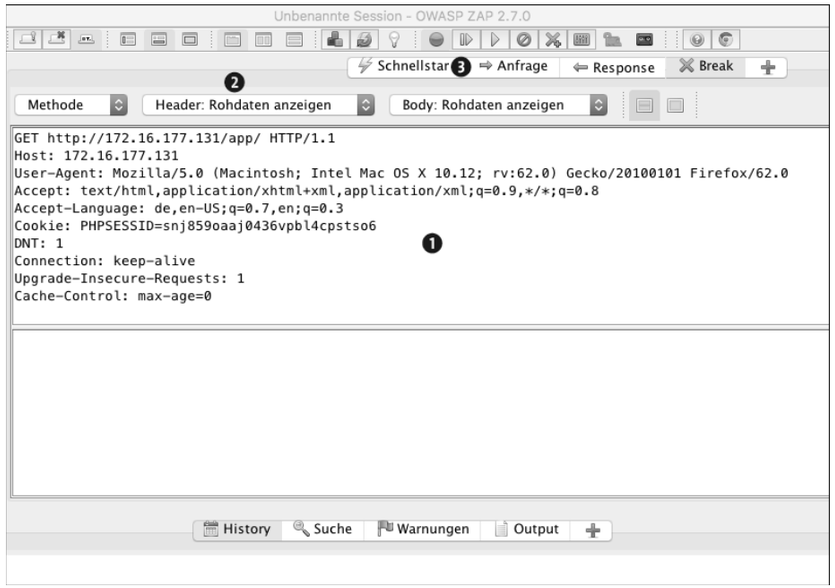

Abbildung 1.5 Der vom ZAP gestoppte Request

Rechts oben sehen Sie jetzt den aufgehaltenen Request (❶ in Abbildung 1.5). Sie können sich den Header auch als Tabelle oder hexadezimal anzeigen lassen ❷, die Tabellenanzeige macht die Manipulation der Header-Werte sehr einfach. Wenn Sie fertig sind, können Sie den Request mit dem Button neben dem für den Breakpoint passieren lassen ❸, der Breakpoint bleibt dabei gesetzt und stoppt die Response des Servers, siehe Abbildung 1.6.

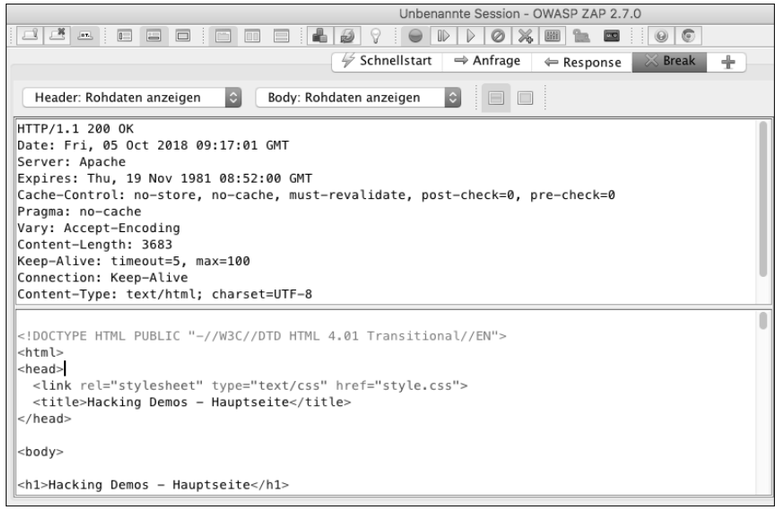

Abbildung 1.6 Die vom ZAP gestoppte Response

Diesmal gibt es außer dem Header auch einen Body. Wenn Sie den Request durchgelassen haben, erscheint er im Browser.

Das war eigentlich schon alles, was Sie über den Proxy wissen müssen. Für alle anderen Funktionen verweise ich auf die Dokumentation – oder Ihre Experimentierfreude; solange Sie sich mit dem ZAP nur an der Demo-Anwendung austoben, kann nichts passieren, diese können Sie bei Bedarf jederzeit entweder neu installieren oder in den Ausgangszustand zurückversetzen.

Alle weiteren Tools brauchen Sie nur in einem Kapitel, und ich habe sie jeweils vor Ort beschrieben. Im Folgenden gibt es nur einen kleinen Überblick.

1.4.2 Browser Exploitation Framework BeEF

Das *Browser Exploitation Framework BeEF* [BeEF] wurde entwickelt, um die Auswirkungen von XSS-Schwachstellen zu demonstrieren. Es läuft auf einem Server, mit dem sich die über XSS übernommenen Clients verbinden. Danach können vom Server aus verschiedene Aktionen auf dem Client gestartet werden. Deren Spannweite reicht vom Auslesen des Clipboards, einem JavaScript-Portscan oder der Installation eines Keyloggers bis zur Ausnutzung von Browser-Schwachstellen zum Öffnen einer Shell.

Das BeEF ist Bestandteil von Kali Linux, und da es auf einem Server laufen muss und das besser nicht derjenige der angegriffenen Webanwendung ist (da sonst die Same-Origin Policy

nicht greift und die Angriffe theoretisch leichter werden), dürfte der Einsatz von Kali Linux die einfachste Lösung sein.

Das BeEF wird in Abschnitt 5.9.8 beschrieben und eingesetzt.

1.4.3 Mini MySqlat0r

Der *Mini MySqlat0r* [Mini_MySqlat0r] ist ein Tool zur Suche nach und Ausnutzung von SQL-Injection-Schwachstellen. Wie der OWASP ZAP ist der Mini MySqlat0r in Java geschrieben. Aber während der ZAP laufend weiterentwickelt wird, ist der Mini MySqlat0r seit 2008 bei Version 0.5 stehengeblieben. Doch er funktioniert immer noch und liefert ganz brauchbare Ergebnisse. Was Sie in Abschnitt 6.6.6 selbst ausprobieren können.

Der Mini MySqlat0r ist nicht in Kali Linux enthalten, als Java-App können Sie ihn aber problemlos auf Ihrem Rechner ausführen.

1.4.4 THC-Hydra

THC-Hydra [THC-Hydra] ist ein Passwortknacker, der Wörterbuch-Angriffe gegen eine Vielzahl von Protokollen durchführen kann. Das Tool ist Bestandteil von Kali Linux, eine Beschreibung finden Sie in Abschnitt 4.16, wo es zum Angriff auf das Login-Formular der Demo-Anwendung zum Einsatz kommt.

1.4.5 Kali Linux

Kali Linux [Kali_Linux] ist eine auf die Suche nach Schwachstellen und ähnliche sicherheitsrelevante Aufgaben spezialisierte Linux-Distribution auf Debian-Basis. Kali Linux enthält alle Programme, die man bei der Suche nach Schwachstellen sowie deren Ausnutzung bzw. Überprüfung brauchen könnte, z. B. den Netzwerkscanner *Nmap* [Nmap] zur Suche nach Hosts und Diensten im lokalen Netz sowie dessen grafische Oberfläche *Zenmap* [Zenmap]. Auch die Web-Schwachstellenscanner *skipfish* [skipfish], *w3af* [w3af] und *Wapiti* [Wapiti] sind enthalten, ebenso *Nikto* [Nikto] zur Suche nach Schwachstellen in Webservern.

Installieren oder als Live-System nutzen

Kali Linux steht in Form von ISO-Images und virtuellen Maschinen zum Download bereit. Es ist sowohl die Installation auf Festplatte oder USB-Stick als auch der Betrieb von DVD oder als virtuelle Maschine möglich. Eine bereits installierte Distribution können Sie wie bei Debian üblich mit dem Dreizeiler

```
apt update
apt dist-upgrade
reboot
```


auf den aktuellen Stand bringen, ebenso können Sie nicht in der Defaultinstallation vorhandene Pakete auf veränderbaren Installationen jederzeit leicht nachinstallieren. Beim Betrieb von DVD funktioniert das natürlich nicht.

Kali Linux nutzen

Laden Sie sich die für Sie passende Variante von *https://www.kali.org* [Kali_Linux] herunter; ich empfehle Ihnen, eine virtuelle Maschine zu nutzen. Egal wie, ich gehe jetzt davon aus, dass Sie Kali Linux gestartet haben.

Da die meisten Tools *root*-Reche benötigen, arbeiten Sie unter Kali Linux per Default als *root*, das zugehörige Passwort lautet »toor«. Die ständige Arbeit als *root* erhöht natürlich die Gefahr, das System zu zerschießen. Also Vorsicht beim rekursiven Löschen von */irgendwas*, ein Leerzeichen hinter dem / sorgt dann dafür, dass Sie die gesamte Installation löschen.

Kali Linux verwendet den GNOME-Windowsmanager. Die gängigsten Tools sind im GNOME-Menü Anwendungen (deutsch) / Applications (englisch) in 14 Hauptrubriken angeordnet. Diese orientieren sich an den verschiedenen Problemstellungen beim Penetration Testing:

- 01 – Informationsbeschaffung / Information Gathering
- 02 – Schwachstellenanalyse / Vulnerability Analysis
- 03 – Webapplikationen / Web Application Analysis
- 04 – Datenbank-Assessment / Database Assessment
- 05 – Passwort-Angriffe / Password Attacks
- 06 – Wireless-Angriffe / Wireless Attacks
- 07 – Reverse Engineering / Reverse Engineering
- 08 – Exploitation-Tools / Exploitation Tools
- 09 – Sniffing & Spoofing / Sniffing & Spoofing
- 10 – Zugang etablieren / Post Exploitations
- 11 – Forensik / Forensics
- 12 – Berichterstellung / Reporting Tools
- 13 – Social Engineering Tools / Social Engineering Tools
- 14 – Systemdienste / System Services

Einige dieser Hauptrubriken sind wiederum in weitere Unterrubriken unterteilt. Das sind z. B. bei der Informationsbeschaffung die Rubriken DNS-Analyse, IDS- & IPS-Identifikation und Live-Host-Erkennung und bei den Webapplikationen die Rubriken CMS & Framework Identification, Webapplikations-Proxies und WebCrawler.

Einige besonders häufig genutzte Tools finden Sie auch direkt auf dem Desktop, darunter Firefox, das Terminal, das Metasploit Framework, das Browser Exploitation Framework (BeEF) und den Texteditor Leafpad.

Die oben vorgestellten Tools finden Sie folgendermaßen:

- OWASP ZAP: unter Anwendungen • Webapplikationen • owasp-zap (siehe Abbildung 1.7)
- Browser Exploitation Framework (BeEF): zum einen wie oben erwähnt auf dem Desktop, zum anderen unter Anwendungen • Exploitation-Tool • beef xss framework
- Mini MySqlatOr: nicht enthalten
- THC-Hydra: unter Anwendungen • Passwort-Angriffe • Online-Angriffe • Hydra

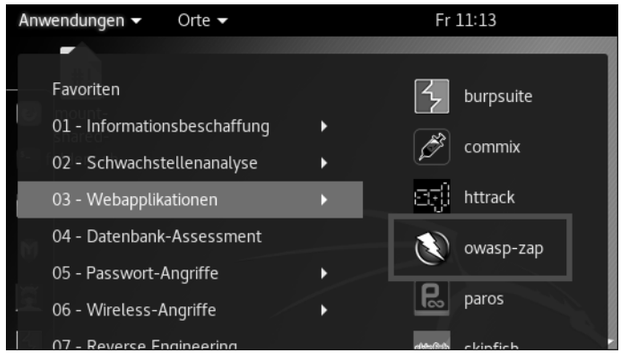

Abbildung 1.7 Der OWASP ZAP im »Anwendungen«-Menü von Kali Linux

1.5 Die Demo-Anwendung

Ich habe eine Beispiel-Webanwendung mit einigen Sicherheitslücken vorbereitet. Sie finden diese App gepackt im Bereich Materialien zum Buch unter *https://www.rheinwerk-verlag.de/youve-been-hacked_4306*.

Ich zitiere von der Startseite der Demo-Anwendung:

> *Dieses Skript ist ziemlich beschränkt: Es können alle Einträge als Titelliste angezeigt werden, und einzelne Einträge können komplett wieder ausgegeben werden. Außerdem können sich die Besucher hier als Benutzer registrieren und registrierte Benutzer sich einloggen.*

Ähnlich sieht es bei den anderen Skripten der Anwendung aus. Die Anwendung dient nämlich nur einem einzigen Zweck: Schwachstellen zu enthalten, damit man sie darin finden, ausnutzen und vorführen kann. Darum gibt es zwar eine »Suchfunktion«, aber sie macht nichts weiter, als den eingegebenen Suchbegriff in einer Meldung, dass die Suche noch nicht implementiert ist, wieder auszugeben:

> *Ihre Suche nach [Suchbegriff] ergab leider kein Ergebnis, da die Suchfunktion noch nicht implementiert wurde!*

Ihr einziger Existenzzweck ist es, eine XSS-Schwachstelle zu enthalten.

Don't try this on the Internet!

Sonst heißt es ja bei gefährlichen Sachen immer, man soll sie nicht zu Hause nachmachen. Aber statt »Don't try this at home« heißt es bei der Demo-Anwendung genau umgekehrt »Only try this at home« (oder im Büro, oder wo auch immer). Aber immer nur in einer geschützten Umgebung, in der kein Cyberkrimineller versuchen kann, die Anwendung anzugreifen. Und niemals auf einem mit dem Internet verbundenen Server. Diese Anwendung dient dazu, den Server darüber zu kompromittieren, und Sie wissen ja: Murphy sorgt schon dafür, dass irgendein Cyberkrimineller gerade zufällig diesen Server scannt und ein paar der Schwachstellen findet!

1.5.1 Systemvoraussetzungen

Die Anforderungen, die die Anwendung an den Server stellt, sind recht gering. Sie ist in PHP geschrieben und läuft auf fast jedem LAMP-System einschließlich XAMPP, egal, ob auf Hardware oder als virtuelle Maschine. Für die Installation müssen lediglich zwei Bedingungen erfüllt sein:

- Die PHP-Version muss kleiner als 7 sein, da die ab Version 7 nicht mehr unterstützte MySQL-Erweiterung verwendet wird.
- Es muss eine MySQL-Datenbank vorhanden sein, in der das Installationsskript die nötigen Tabellen anlegen und Daten eintragen kann.

Damit File-Inclusion-Schwachstellen vorhanden sind, müssen Sie PHP unsicher konfigurieren. Ist das nicht möglich, fallen aber nur diese Schwachstellen weg, der Rest der Anwendung funktioniert trotzdem wie vorgesehen.

1.5.2 Die Installation der Demo-Anwendung allgemein

Sie finde die Beispiel-Webanwendung im Bereich Materialien zum Buch unter *https://www.rheinwerk-verlag.de/youve-been-hacked_4306*.

Laden Sie dieses Archiv herunter und entpacken Sie es. Nach dem Entpacken erhalten Sie ein neues Unterverzeichnis *ins-webroot* mit folgender Verzeichnisstruktur:

- */hacking-index.html* als Startseite
- */app* mit der Demo-Anwendung
- */daten* mit den weiteren Daten zum Buch, z. B. Vorlagen für Tests und Angriffe, Beispielexploits und ähnliches

Diese drei Objekte müssen Sie in das Webroot-Verzeichnis Ihres Webservers kopieren. Wenn Sie über keinen eigenen Webserver als Spielwiese verfügen: keine Sorge! In Abschnitt 1.5.3 zeige ich Ihnen, wie Sie einen Webserver in einer virtuellen Maschine starten.

Nach dem Kopieren können Sie sowohl die Demo-Anwendung als auch die zusätzlichen Informationen zum Buch wie Angriffe, Exploits und ähnliches über *http://[Ihr-Webserver]/hacking-index.html* aufrufen.

Wechseln Sie nach dem Kopieren im Webroot-Verzeichnis in das Verzeichnis */app/lib/*, und öffnen Sie die Datei *config.inc* in einem Texteditor. Diese Datei sieht jetzt aus wie in Listing 1.1.

```
<?php

/*
     Konfigurationsdatei fuer Hacking-Demo
*/



/*
Die MySQL-Parameter:
*/
define ("USERNAME","hackdemo");  // Benutzername
define ("PASSWORD","hackdemo");  // Passwort
define ("HOSTNAME","localhost"); // Hostname des MySQL-Servers
define ("DATENBANK","hackdemo"); // Name der Datenbank



/*
Default-Spracheinstellung - Moegliche Werte: de oder en
*/
define ("DEFAULTSPRACHE","de");



/*
Vollstaendiger Pfad zum Avatarverzeichnis /app/avatare/
*/
define ("AVATARPFAD","/var/www/app/avatare/");

?>
```

Listing 1.1 Die Konfigurationsdatei »/app/lib/config.inc« der Demo-Anwendung

Hier müssen Sie an zwei Stellen Änderungen vornehmen:

1. Im Bereich mit den MySQL-Parametern tragen Sie die Daten der für die Demo-Anwendung vorgesehenen MySQL-Datenbank ein: Benutzername und Passwort, den Hostname des MySQL-Servers (meist »localhost«, weil er auf dem gleichen Server wie der Webserver läuft) und den Namen der Datenbank.
2. Unten geben Sie den vollständigen Pfad zum Avatarverzeichnis */app/avatare/* ein. Ist das Webroot-Verzeichnis */var/www/*, lautet der vollständige Pfad also wie im Beispiel in Listing 1.1 */var/www/app/avatare/*. Dieses Verzeichnis muss für die Webanwendung beschreibbar sein.

Danach müssen Sie nur noch das Installationsskript */app/install.php* starten, das die nötigen Tabellen in der Datenbank anlegt und einige erste Einträge vornimmt, danach können Sie die Demo-Anwendung aufrufen und mit der Schwachstellensuche loslegen.

1.5.3 Die Installation in einer Linux-VM

In diesem Abschnitt gebe ich Ihnen eine ausführlichere Anleitung zur Installation der Demo-Anwendung in einer virtuellen Maschine mit LAMP-Stack. Als Beispiel dient mir der TurnKey LAMP Stack [Turnkey-VM]. Laden Sie sich die zu Ihrer Virtualisierungslösung passende Datei der Version 14.2 von *https://www.turnkeylinux.org/lamp* herunter (ab Version 15.0 kommt PHP 7.0 zum Einsatz, und das können wir nicht brauchen).

Sie können beispielsweise VirtualBox oder VMware vSphere für die Virtualisierung nutzen. Bitte informieren Sie sich in der entsprechenden Dokumentation, welche Einstellungen Sie genau benötigen. Kurz: Sie brauchen eine virtuelle Maschine der x64-Architektur und sollten sicherheitshalber 2 GB Arbeitsspeicher und 4 GB Festplattenplatz reservieren. Wenn diese Voraussetzungen gegeben sind, starten Sie die heruntergeladene VM.

Linux konfigurieren

Nachdem das Linux in der VM gestartet ist, müssen Sie es konfigurieren. Dazu fragt das System einige Informationen ab:

- das Passwort für den Linux-Account *root*, siehe Abbildung 1.8
- das Passwort für den MySQL-Account *root*, siehe Abbildung 1.9
- Die Initialisierung der TurnKey Hub Services überspringen Sie (siehe Abbildung 1.10, mit der Taste ⇆ springen Sie erst zu den Buttons und danach zu Skip, ↵ schickt das Formular ab).
- Das Gleiche machen Sie mit den System Notifications and Critical Security Alerts (Abbildung 1.11) und der sofortigen Installation der Sicherheitsupdates (Abbildung 1.12).

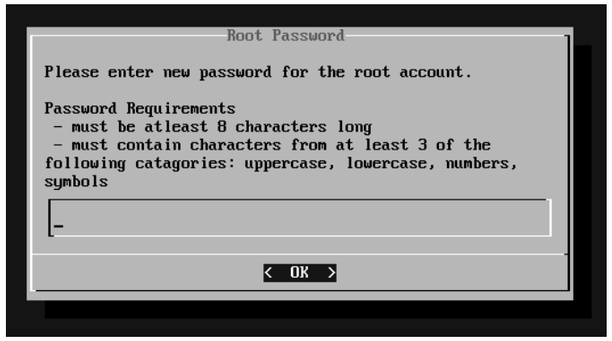

Abbildung 1.8 Sie müssen ein neues Passwort für den »root«-Account von Linux eingeben.

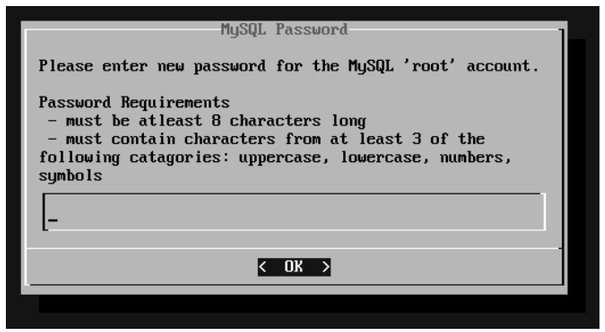

Abbildung 1.9 Auch MySQL hat einen »root«-Account, auch der braucht ein neues Passwort.

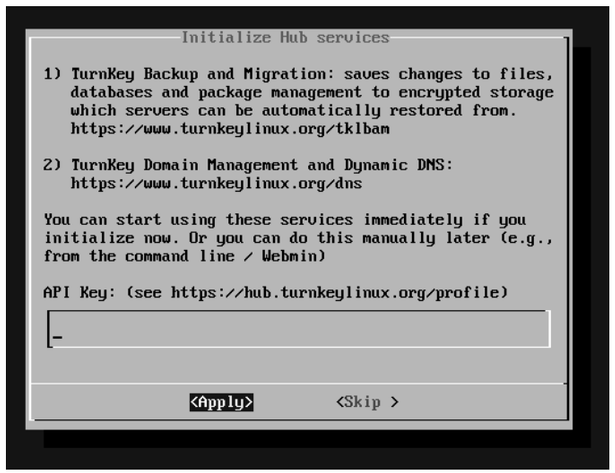

Abbildung 1.10 Die Initialisierung der TurnKey Hub Services überspringen Sie.

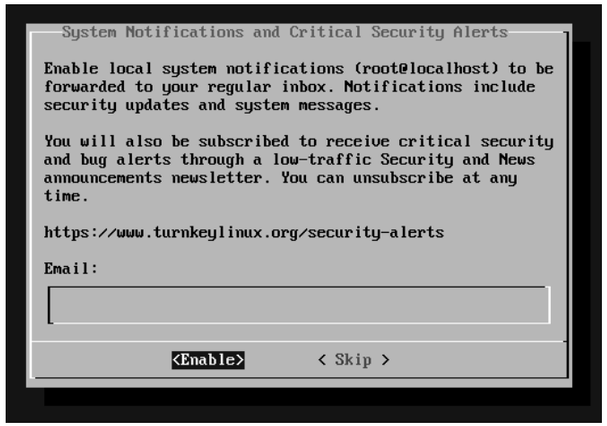

Abbildung 1.11 Auch die Konfiguration der »System Notifications and Critical Security Alerts« überspringen Sie.

Abbildung 1.12 Sicherheitsupdates müssen Sie jetzt nicht installieren, die Demo-Anwendung ist sowieso unsicher.

Danach ist die Installation abgeschlossen, und Sie können den LAMP-Stack nutzen. Seine IP-Adresse sowie die Adressen einiger Tools zeigt er Ihnen freundlicherweise an, siehe Abbildung 1.13. Rufen Sie den Webserver mit seiner IP-Adresse auf, und wählen Sie auf der Startseite (siehe Abbildung 1.14) die webbasierte Administrationsoberfläche Webmin aus.

Jetzt gibt es eine Fehlermeldung: Die VM nutzt ein selbst signiertes Zertifikat für die HTTPS-Verbindung (Abbildung 1.15), und das müssen Sie erst mit einer Ausnahmegenehmigung zulassen (Abbildung 1.16), bevor die Seite geladen wird. Danach können Sie sich als Benutzer *root* mit dem eben vergebenen Passwort bei Webmin einloggen (Abbildung 1.17).

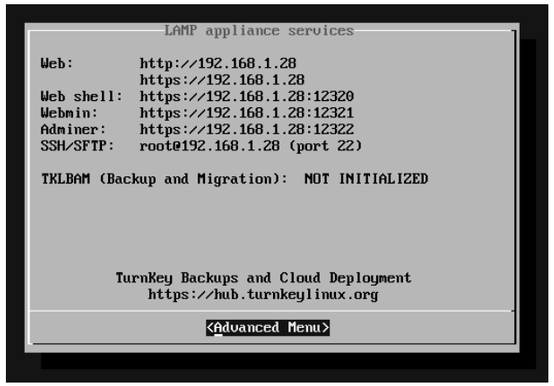

Abbildung 1.13 Die IP-Adresse der LAMP-Appliance und ihrer Konfigurationstools

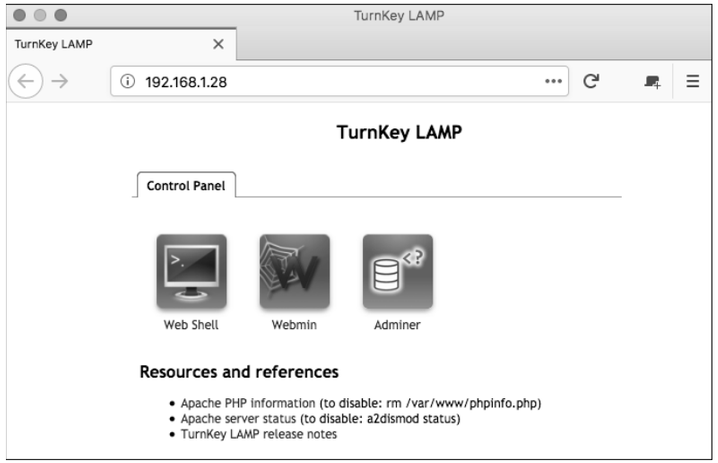

Abbildung 1.14 Die Startseite des Webservers der VM

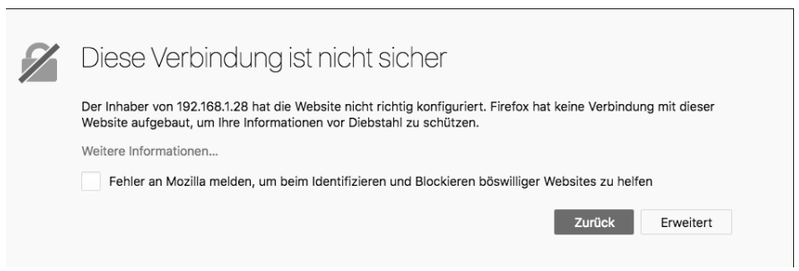

Abbildung 1.15 Selbst signierte Zertifikate mögen die Browser, hier Firefox, nicht. Prinzipiell zu recht, aber in diesem Fall ...

Abbildung 1.16 ... ist eine Ausnahme angebracht.

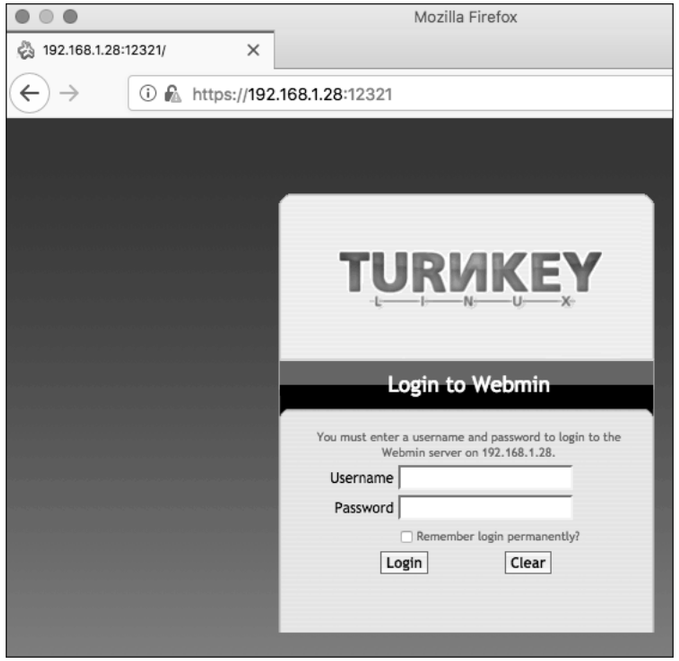

Abbildung 1.17 Das Login-Formular der webbasierten Administrationsoberfläche Webmin

PHP konfigurieren

Wählen Sie aus der angezeigten Auswahl die PHP Configuration aus (Abbildung 1.18). Sie müssen die Konfiguration für `mod_php` ändern, dazu rufen Sie dahinter Manage auf (Abbildung 1.19). Wenn Sie schon mal dabei sind, können Sie PHP gleich generell unsicher konfigurieren, frei nach dem Motto »Wenn schon, denn schon«.

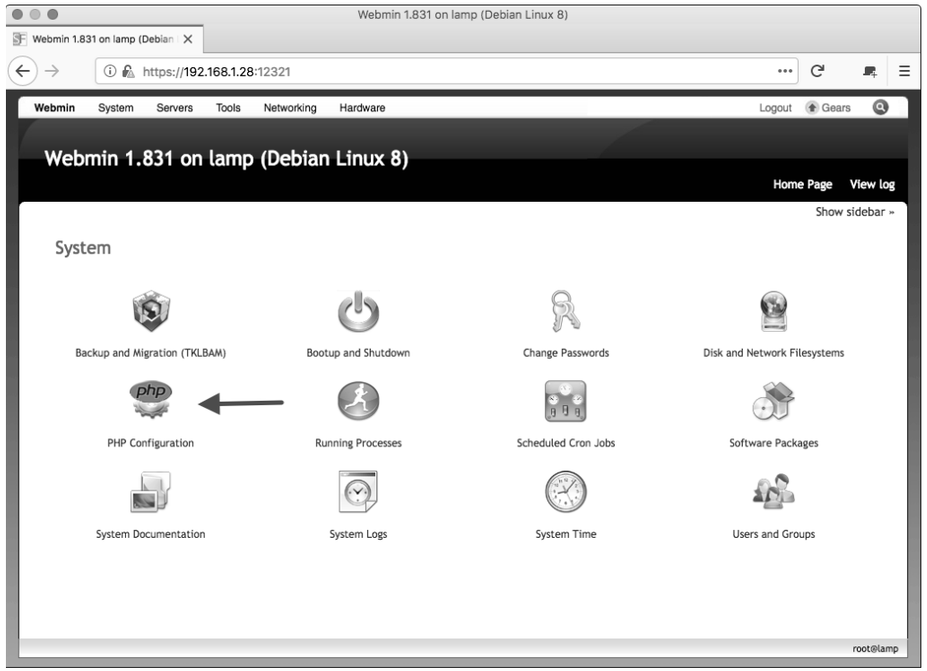

Abbildung 1.18 Die Systemkonfiguration von Webmin

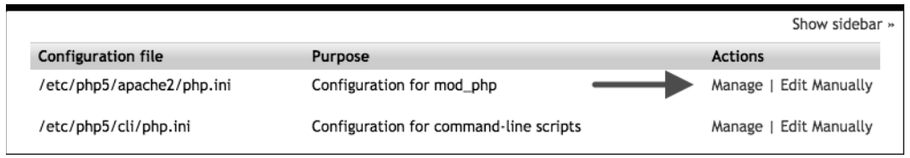

Abbildung 1.19 Auswahl der Konfiguration von »mod_php«

Fangen Sie mit den PHP Variables an (Abbildung 1.20), und schalten Sie alles, was die Sicherheit erhöhen soll, aus:

- Quote all input variables? – No
- Quote data generated at runtime? – No

- Turn all input into global variables? – Yes
- Create old-style arrays like HTTP_GET_VARS? – Yes

Lediglich der Punkt Turn command-line parameters into global variables? ist egal, da Sie keine Kommandozeilenparameter benötigen. Jetzt müssen Sie noch die Einstellungen, die wie in Abbildung 1.21 aussehen sollten, mit Save sichern, dann kann es weitergehen.

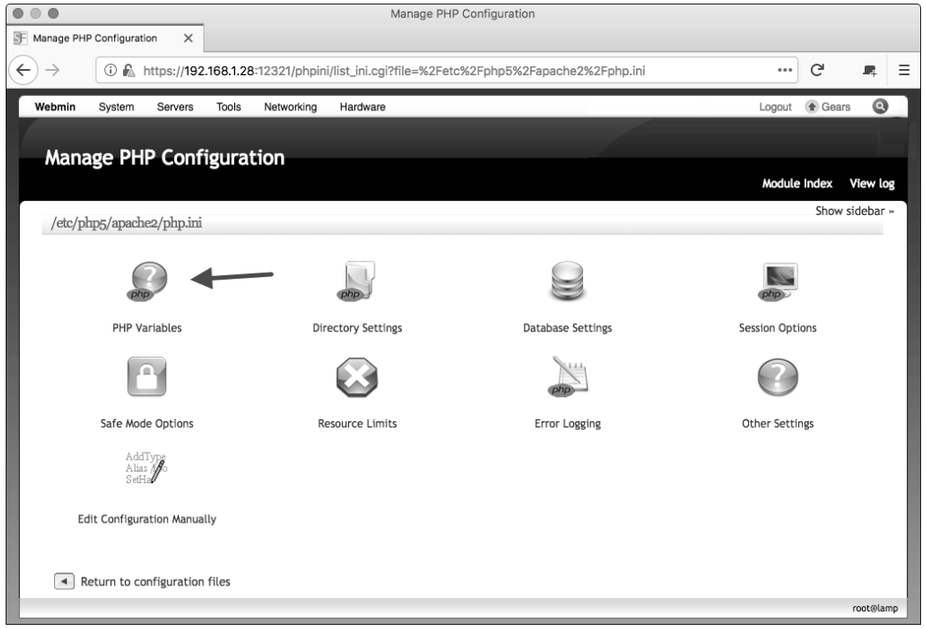

Abbildung 1.20 Die PHP-Konfiguration von Webmin

Abbildung 1.21 So sollten die PHP-Variablen konfiguriert sein.

Als Nächstes sind die Directory Settings an der Reihe. Hier müssen Sie nichts ändern; das von der Demo-Anwendung benötigte Heraufladen von Dateien (Allow file uploads?) ist

bereits eingeschaltet (Abbildung 1.22). Auch in den DATABASE SETTINGS und den SESSION OPTIONS müssen Sie nichts ändern.

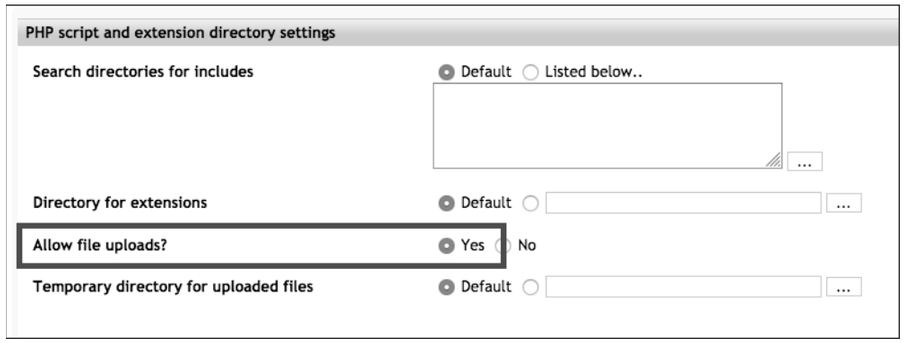

Abbildung 1.22 Die »Directory Settings« sind bereits richtig konfiguriert.

In den SAFE MODE OPTIONS müssen Sie den Safe Mode ausschalten (ENABLE SAFE MODE? – NO, siehe Abbildung 1.23).

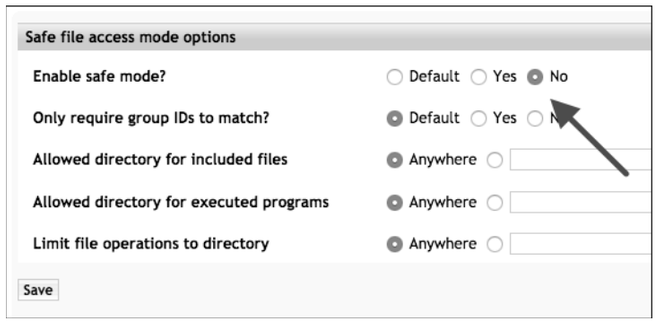

Abbildung 1.23 Wenn PHP unsicher sein soll, müssen Sie den Safe Mode natürlich ausschalten.

In den RESOURCE LIMITS müssen Sie nichts ändern, ebenso wenig beim ERROR LOGGING. Auf diese Einstellungen werden Sie aber eventuell nach den Tests zurückkommen, wenn Sie das in Abschnitt 1.6 beschriebene Problem der unzureichenden Protokollierung korrigieren möchten.

Unter OTHER SETTINGS sind wieder Änderungen nötig:

- ALLOW PHP SCRIPTS STARTING WITH <? ? – YES
- ALLOW OPENING OF REMOTE INCLUDES? – YES

Das ebenfalls nötige ALLOW OPENING OF URLS AS FILES? ist schon eingeschaltet.

Nachdem Sie diese Einstellungen, die wie in Abbildung 1.24 aussehen, gesichert haben, sind Sie mit der PHP-Konfiguration fertig.

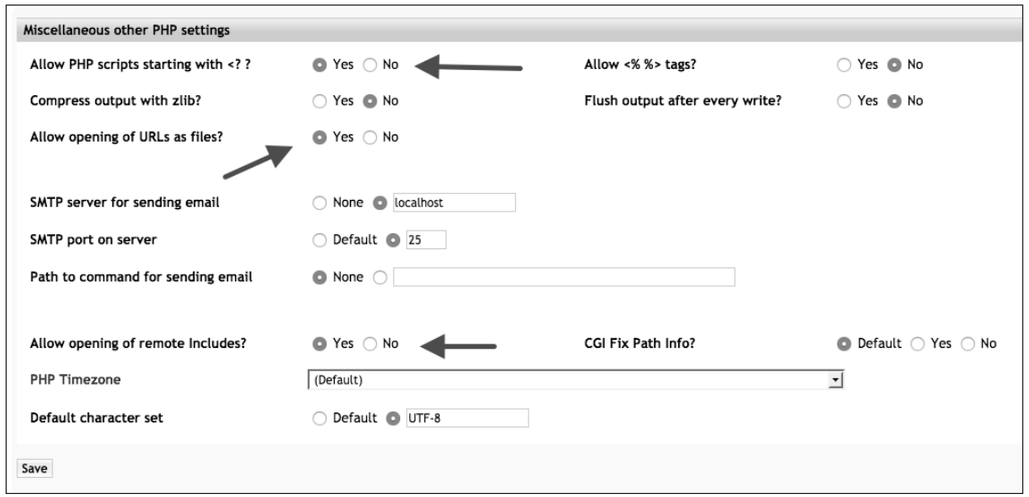

Abbildung 1.24 Einige weitere PHP-Einstellungen

MySQL konfigurieren

Jetzt ist MySQL an der Reihe. Wählen Sie im Menü SERVERS den MYSQL DATABASE SERVER (siehe Abbildung 1.25), und loggen Sie sich mit dem vorhin gesetzten MySQL-*root*-Passwort ein. Jetzt müssen Sie MySQL so konfigurieren, dass Sie danach die Konfigurationsdatei der Demo-Anwendung aus Listing 1.1 konfigurieren können. Zur Erinnerung: Es ging um folgende Einstellungen:

```
define ("USERNAME","hackdemo");   // Benutzername
define ("PASSWORD","hackdemo");   // Passwort
define ("HOSTNAME","localhost");  // Hostname des MySQL-Servers
define ("DATENBANK","hackdemo");  // Name der Datenbank
```

Der Hostname LOCALHOST stimmt schon mal, das ist die Defaulteinstellung. Jetzt brauchen Sie zuerst einen Benutzer und ein Passwort. Wählen Sie dazu die USER PERMISSIONS aus (❶ in Abbildung 1.26), danach in der oberen Reihe CREATE NEW USER (Abbildung 1.27). Ich habe den Benutzer in Abbildung 1.28 wie in der *config.inc* `hackdemo` genannt, und auch das Passwort ist wie dort `hackdemo`. Wählen Sie alle PERMISSIONS aus – wenn schon, dann soll ein Angreifer auch seinen Spaß haben. Jetzt können Sie den neuen Benutzer mit CREATE anlegen.

Kehren Sie nun zur Datenbankliste zurück (der Link ist ganz unten auf der Seite, eventuell müssen Sie etwas nach unten scrollen). Jetzt benötigen Sie noch eine Datenbank für die Demo-Anwendung. Dazu wählen Sie in der obersten Zeile CREATE A NEW DATABASE aus (❷ in Abbildung 1.26). Als Name habe ich in Abbildung 1.29 wie in der Konfigurationsdatei `hackdemo` genommen. Mehr müssen Sie nicht konfigurieren, das übernimmt das Installationsskript der Demo-Anwendung. Legen Sie die Datenbank mit CREATE an, schon sind Sie hier fertig.

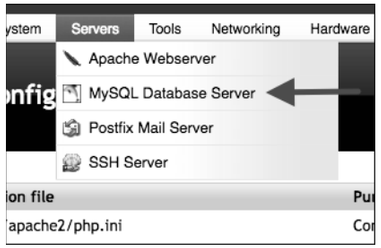

Abbildung 1.25 Auf dem Weg zum MySQL Database Server

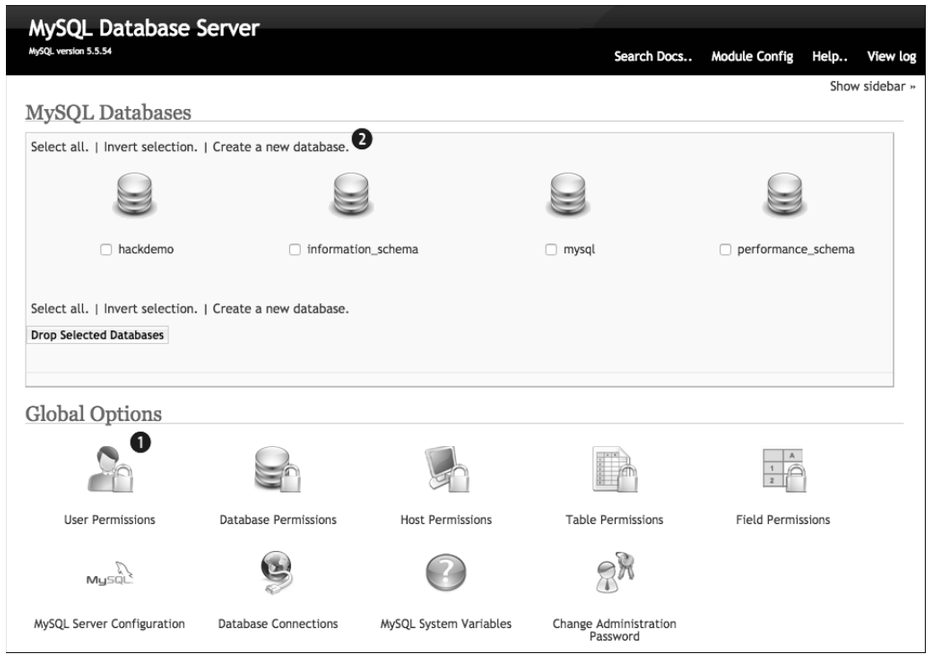

Abbildung 1.26 Die Startseite der MySQL-Administration

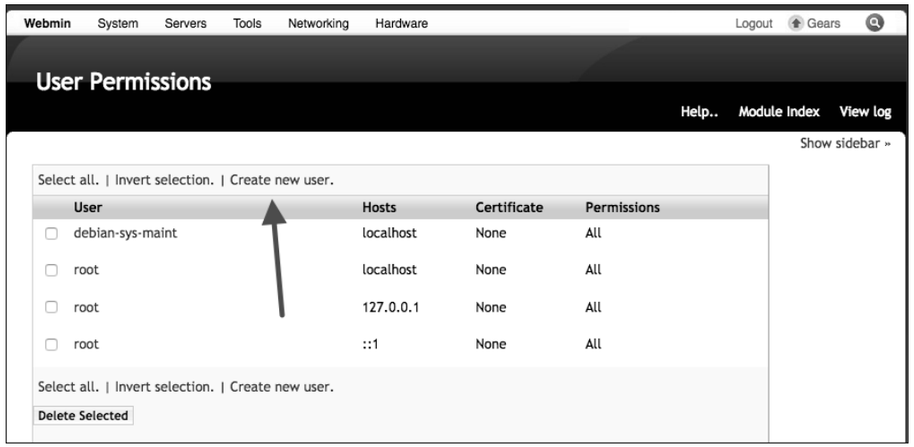

Abbildung 1.27 Die »User Permissions« von MySQL; hier werden auch neue Benutzer angelegt.

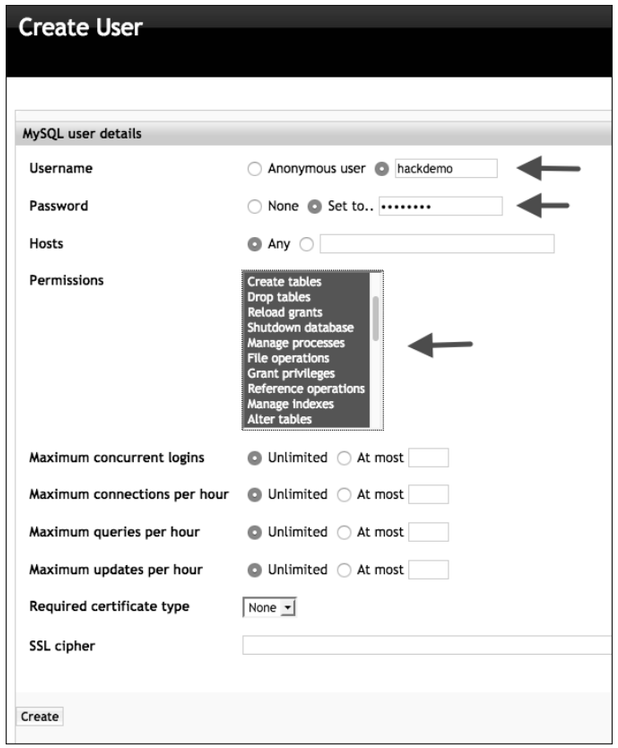

Abbildung 1.28 Der neue Benutzer – hier wird die Datenbank an die Anwendung angepasst, nicht wie sonst üblich umgekehrt. ☺

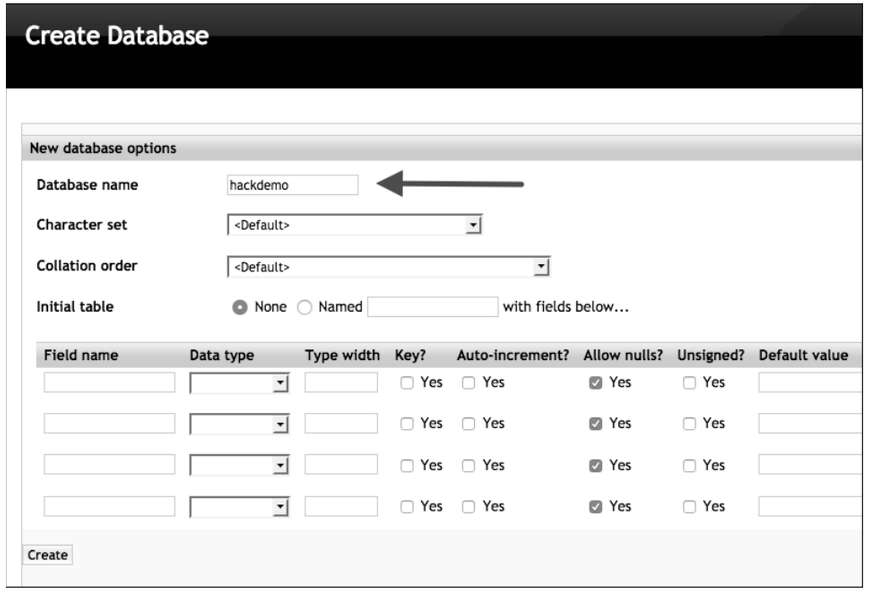

Abbildung 1.29 Die neue Datenbank, ebenfalls angepasst an die Demo-Anwendung

Demo-Anwendung installieren

Wählen Sie nun unter TOOLS den FILE MANAGER aus (Abbildung 1.30), und wechseln Sie dort zuerst in */var* und dann in das sogenannte *Webroot-Verzeichnis /var/www* (Abbildung 1.31). Alles, was Sie darunter speichern, ist später über den Webserver erreichbar.

Da Sie öfter hierhin müssen, sollten Sie für das Verzeichnis ein Bookmark anlegen (❶ in Abbildung 1.31), danach wählen Sie den Datei-Upload (❷ in Abbildung 1.31) und laden das Archiv der Demo-Anwendung hoch (Abbildung 1.32).

Abbildung 1.30 Auf zum File Manager!

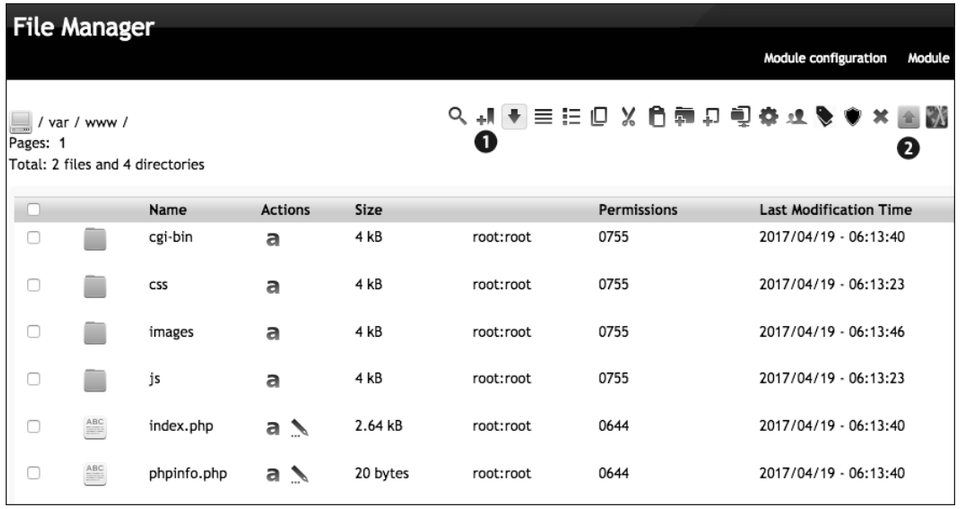

Abbildung 1.31 Das Webroot-Verzeichnis »/var/www« im File Manager

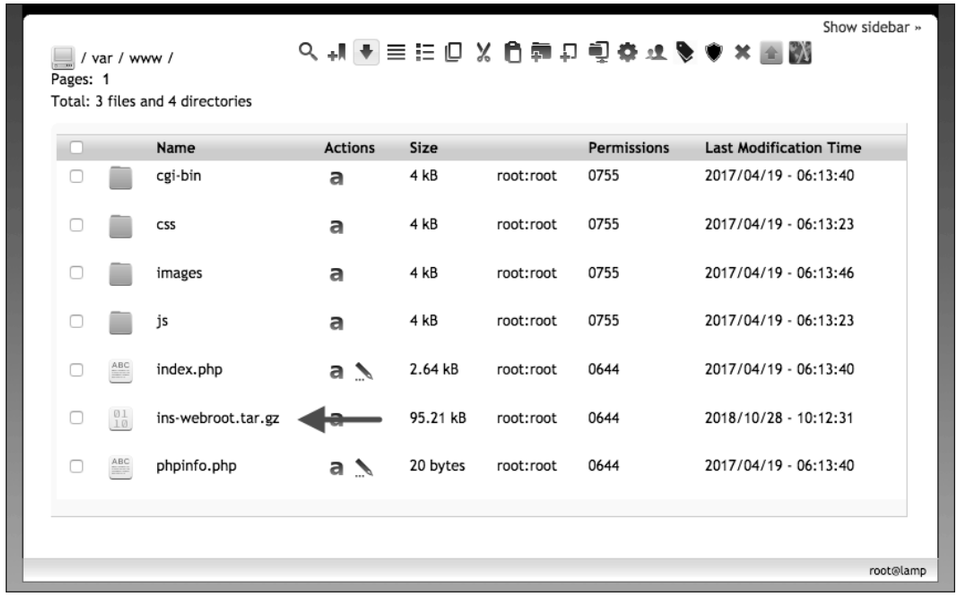

Abbildung 1.32 Das Archiv mit der Demo-Anwendung im Webroot-Verzeichnis

Wählen Sie danach unter Tools die Command Shell (Abbildung 1.33), und führen Sie folgende Befehle aus (Abbildung 1.34):

- Wechseln Sie in das Verzeichnis */var/www*: `cd /var/www`
- Entpacken Sie das Archiv: `tar -xf ins-webroot.tar.gz`
- Verschieben Sie die Verzeichnisse und Dateien aus dem beim Entpacken erzeugten Verzeichnis in das Webroot-Verzeichnis: `mv ins-webroot/* .`
- Löschen Sie das nun leere Vezeichnis: `rmdir ins-webroot`

Abbildung 1.33 Shell aufrufen ...

Abbildung 1.34 ... und Befehle eingeben

Ab jetzt können Sie sowohl die Demo-Anwendung als auch die zusätzlichen Informationen zum Buch wie Angriffe, Exploits und ähnliches über *http://[IP-Adresse der Virtuellen Maschine]/hacking-index.html* aufrufen. Beim direkten Aufruf der VM ohne Pfad erscheint nach wie vor das Menü der TurnKey-Weboberfläche.

Jetzt rufen Sie wieder den File Manager auf und wählen das Bookmark */var/www* (Abbildung 1.35). Hier gibt es nun das Verzeichnis *app*. Öffnen Sie erst dieses Verzeichnis und danach darin das Verzeichnis *lib*. Öffnen Sie die Datei *config.inc* (Abbildung 1.36) im Editor, und passen Sie die Konfiguration an. Den Pfad zum Avatarverzeichnis müssen Sie nicht ändern, er passt bereits. Nur die Datenbankkonfiguration müssen Sie an Ihre eben getroffenen Einstellungen anpassen, es sei denn, Sie haben so wie ich die Defaultwerte der Demo-Anwendung für die Konfiguration der MySQL-Datenbank verwendet, dann müssen Sie hier gar nichts tun.

Wenn Sie die Konfigurationsdatei ändern, legen Sie bitte zwei Sicherheitskopien an: eine direkt im Verzeichnis */app/lib*, so dass Sie eine überschriebene oder anderweitig beschädigte Datei austauschen können, und eine in einem anderen Verzeichnis, z. B. dem Home-Verzeichnis des *root*-Benutzers. Wenn Sie die Demo-Anwendung so zerschossen haben, dass Sie das komplette */app*-Verzeichnis ersetzen müssen, können Sie auf diese Kopie zurückgreifen.

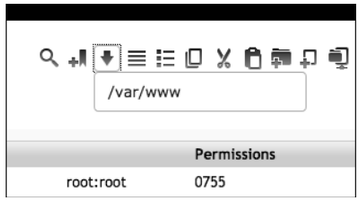

Abbildung 1.35 Das Bookmark zum Webroot-Verzeichnis »/var/www«

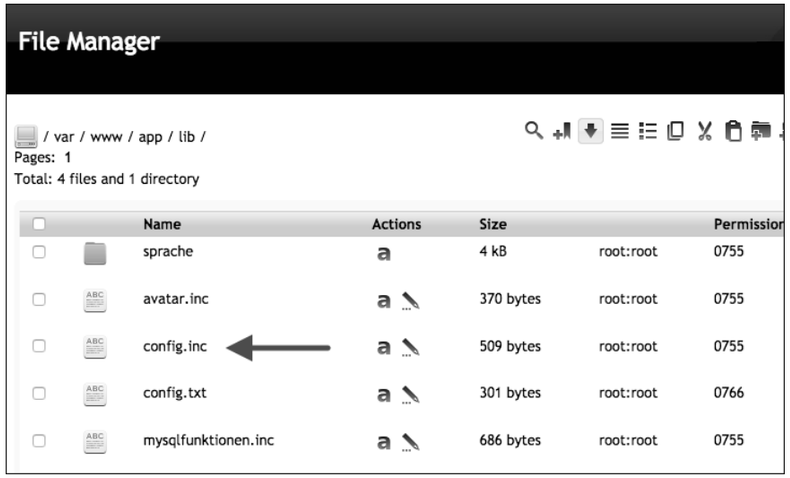

Abbildung 1.36 Das Verzeichnis »/var/www/app/lib« mit der Konfigurationsdatei der Demo-Anwendung

Jetzt ist der Webbrowser an der Reihe. Rufen Sie dessen Startseite auf, und ergänzen Sie die URL um `/app`. Die Demo-Anwendung läuft, gibt aber eine SQL-Fehlermeldung aus, da sie noch nicht vollständig installiert ist (Abbildung 1.37):

```
1146: Table 'hackdemo.text_tabelle' doesn't exist
```

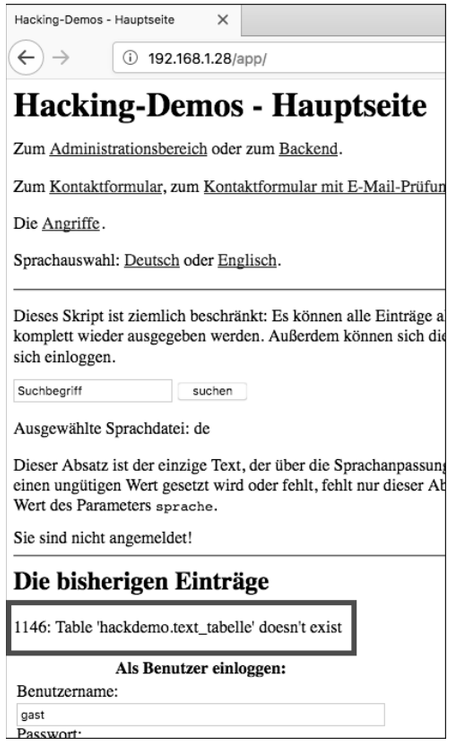

Abbildung 1.37 Die Demo-Anwendung, bevor das Installationsskript gelaufen ist: Es gibt einen SQL-Fehler.

Dazu rufen Sie das Skript *<IP-Adresse>/app/install.php* auf. Es läuft ohne Probleme durch, und danach ist die Demo-Anwendung vollständig installiert. Wie am Ende der Seite schon steht (Abbildung 1.38):

SIE KÖNNEN DIE DEMO JETZT STARTEN!

Und die sieht dann wie in Abbildung 1.39 aus. Jetzt können Sie mit den Tests starten!

Auf der Startseite ist unter DIE ANGRIFFE die Startseite der Zusatzinformationen verlinkt. Hier finden Sie einen Teil der Testeingaben und Angriffe, so dass Sie die nicht aus dem Buch abtippen müssen, sondern einfach kopieren können.

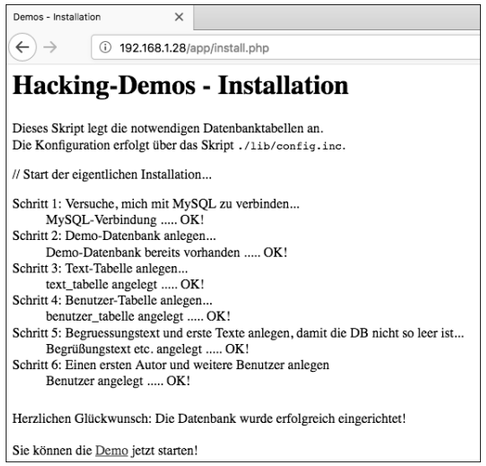

Abbildung 1.38 Die Installation war erfolgreich ...

Abbildung 1.39 ... die Demo-Anwendung kann gestartet werden.

Wenn was schiefgeht: Aufräumen!

Wenn Sie die Demo-Anwendung durch irgendeinen Test so sehr verunstaltet haben, dass Sie damit nicht mehr weiterarbeiten möchten, können Sie sie relativ leicht wieder in Ordnung bringen.

Wenn die Datenbank mit Einträgen oder Benutzern überfüllt ist

1. Gehen Sie in Webmin zur Konfigurationsseite des MySQL Servers. Sie enthält jetzt auch die Datenbank *hackdemo* und sieht wie in Abbildung 1.40 aus.
2. Wählen Sie die Datenbank *hackdemo* aus. Sie sehen nun die vorhandenen Tabellen.
3. Wählen Sie durch Klick auf SELECT ALL alle Tabellen aus (Abbildung 1.41).
4. Nun können Sie durch Klick auf DROP SELECTED OBJECTS alle Tabellen auf einmal löschen.
5. Nachdem die Tabellen gelöscht wurden, können Sie sie durch den Aufruf des Installationsskripts */app/install.php* wieder anlegen lassen. Die Datenbank befindet sich danach wieder im gleichen Zustand wie direkt nach der Installation.

Sie können auch nur einzelne Tabellen löschen – das Installationsskript prüft, ob eine Tabelle vorhanden ist, und legt nur diejenigen an, die fehlen.

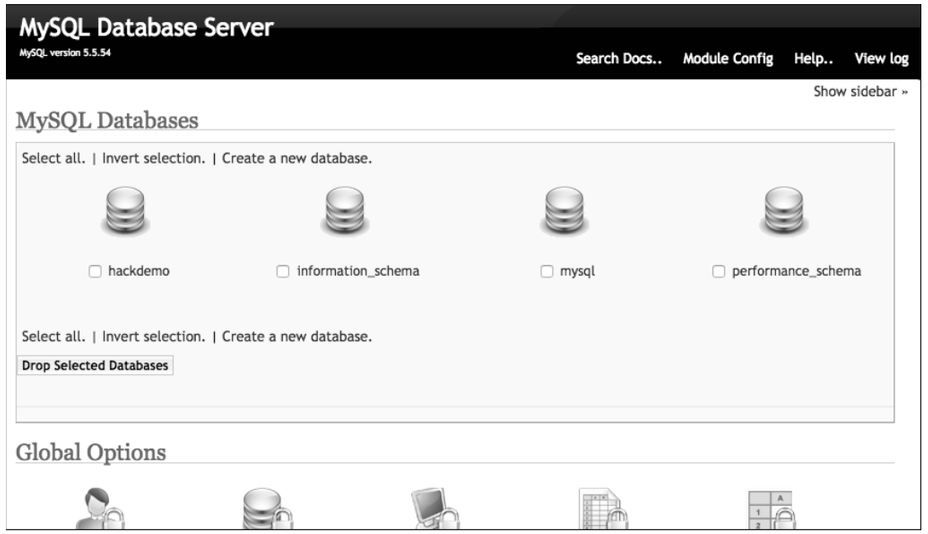

Abbildung 1.40 Die Startseite der MySQL-Administration mit der »hackdemo«-Datenbank

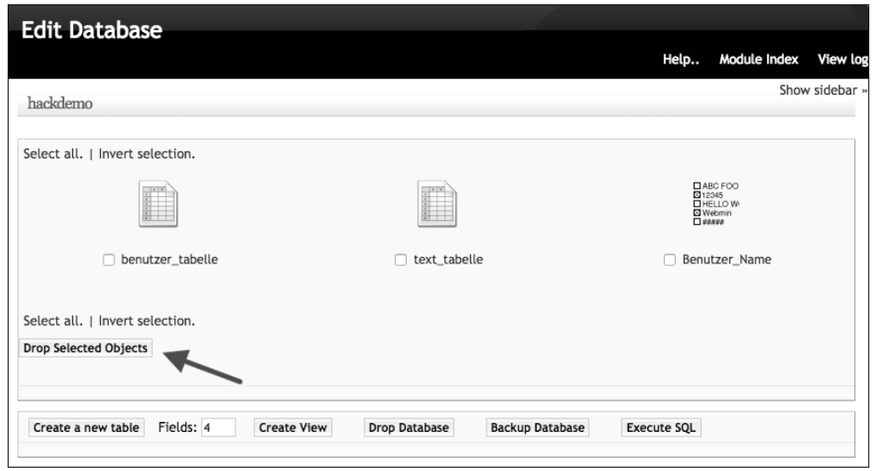

Abbildung 1.41 Die Tabellen der Datenbank »hackdemo«: Mit »Select all« wählen Sie alle auf einmal aus.

Abbildung 1.42 Danach werden die ausgewählten Tabellen gelöscht.

Wenn Sie Dateien gelöscht haben

Zunächst einmal können Sie das Archiv der Demo-Anwendung jederzeit in ein anderes Verzeichnis entpacken und dann einzelne Dateien in der installierten Anwendung austauschen.

Von kritischen Dateien gibt es auch immer Sicherheitskopien im jeweiligen Verzeichnis, z. B. für die HTTP Basic Authentication.

1.6 OWASP Top 10 Platz 10: Insufficient Logging & Monitoring

Insufficient Logging & Monitoring, also die unzureichende Protokollierung und Überwachung, fällt wie oben schon erwähnt etwas aus dem üblichen Muster der Top-10-Einträge, bei denen

es sich sonst um direkt ausnutzbare Schwachstellen in bzw. Angriffe auf Webanwendungen handelt. Eine unzureichende Protokollierung und Überwachung lässt sich dagegen generell nicht für Angriffe ausnutzen.

Ob eine Anwendung oder ein Server irgendetwas protokolliert, ist für einen Angreifer zunächst einmal völlig uninteressant, ebenso ob irgendjemand die protokollierten Daten auswertet oder nicht. Es gibt keine Angriffstechnik, die über eine fehlende Protokollierung die Kompromittierung des Servers erlaubt. Und auch keine Möglichkeit, eine unterlassene Überwachung der Protokolle für Angriffe auf die Benutzer zu nutzen. Oder irgendeine Kombination davon.

Das Einzige, was es bisher gab, waren direkte Angriffe über Logfiles: Wenn eine XSS-Schwachstelle das Einschleusen von JavaScript-Schadcode in die Logfiles erlaubte *und* der Administrator die Logfiles mit einem Tool auswertete, dass JavaScript ausführt, war darüber ein Angriff möglich, z. B. um die Webanwendung im Namen des Administrators zu manipulieren oder durch eine Drive-by-Infektion dessen Rechner mit Schadcode zu infizieren.

1.6.1 Auch Nichtstun kann gefährlich sein

Und trotzdem hat es das Insufficient Logging & Monitoring in die Top 10 geschafft, wenn auch »nur« auf Platz 10, während ein tatsächlicher Angriff wie die Cross-Site-Request-Forgery, mit der sich Schaden anrichten lässt und auch Schaden angerichtet wurde, auf Platz 13 steht [OWASP_Top10-2017-RC2]. Dies hat zwei Gründe: Erstens, dass CSRF-Schwachstellen nur noch in ca. 5 % der Anwendungen gefunden wurden, da die meisten Webanwendungen inzwischen mit Frameworks entwickelt werden und diese im Allgemeinen einen CSRF-Schutz enthalten. Und zweitens, dass sich das Insufficient Logging & Monitoring zwar nicht direkt für Angriffe ausnutzen lässt, jedoch erheblich dazu beiträgt, dass stattfindende Angriffe nicht erkannt werden. Und damit spielt es den Angreifern eben doch in die Hände.

Wie sehr, zeigt mitunter schon ein Penetrationstest. Die Aktionen des Testers sollten so umfangreich protokolliert werden, dass sich daraus der Angriff und die Folgen nachvollziehen lassen. Ist dies nicht geschehen, haben Sie im Ernstfall ein Problem: Sie können einen Angriff entweder gar nicht erkennen oder seine Folgen nicht korrekt nachvollziehen.

Daher hier ein Vorgriff auf die späteren Tests der Demo-Anwendung: Achten Sie doch einmal darauf, ob Sie Ihre Aktionen überhaupt im Logfile des Web- und Datenbankservers finden, und wenn ja, ob Sie anhand der Aufzeichnungen den »Angriff« nachvollziehen können, denn die Webanwendung selbst protokolliert gar nichts.

Das »Nichterkennen« ist auch schon dann ein Problem, wenn es sich nur um Angriffsversuche und nicht um erfolgreiche Angriffe handelt: Die meisten Angriffe beginnen mit der Suche nach einer Schwachstelle. Werden diese Angriffsversuche nicht erkannt und danach unterbunden, führen sie den Angreifer irgendwann zum Ziel. 2016 dauerte es durchschnittlich 191 Tage, bis ein erfolgreicher Angriff (*Data Breach*) erkannt wurde, und der Angriff selbst

dauerte im Durchschnitt 66 Tage [CostBreach]. Beides ist mehr als genug Zeit für den Angreifer, um großen Schaden anzurichten.

1.6.2 Wann ist eine Anwendung betroffen?

Eine unzureichende Protokollierung, die mangelhafte Erkennung von sicherheitsrelevanten Ereignissen sowie eine unzureichende Überwachung und Reaktion können an vielen Stellen auftreten:

- Prüfwürdige Ereignisse wie Logins, fehlgeschlagene Logins und hochwertige Transaktionen werden nicht protokolliert.
- Warnungen und Fehler erzeugen gar keine, unzureichende oder unklare Logfile-Einträge.
- Die Logfiles von Anwendungen und APIs werden nicht auf verdächtige Aktivitäten überwacht.
- Logfiles werden nur lokal gespeichert, so dass keine zentrale Auswertung möglich ist und sie außerdem vom Angreifer manipuliert werden können.
- Es gibt keine angemessenen Warnschwellen und keinen angemessenen Eskalationsprozess, oder die vorhandenen sind ineffektiv.
- Penetrationstests und Scans mit Sicherheitstools (*Dynamic Application Security Testing*, DAST) wie dem OWASP Zed Attack Proxy [OWASP_ZAP] erzeugen keine Alarmmeldungen. Das bedeutet, dass auch ein echter Angriff keinen Alarm auslösen würde.
- Die Anwendung ist nicht in der Lage, aktive Angriffe in Echtzeit oder zumindest nahezu in Echtzeit zu erkennen und zu eskalieren oder davor zu warnen.

Selbst wenn alle diese Fälle in einer Anwendung bzw. Installation nicht auftreten, gibt es noch einen Fallstrick: Wenn Logging- und Alarmierungsevents für den Benutzer und damit auch einem potentiellen Angreifer sichtbar sind, erleichtern sie dem Angreifer unter Umständen die Arbeit. Die Anwendung besitzt dann ein Informationsleck, womit wir wieder bei Platz 3 der OWASP Top 10, »Sensitive Information Exposure«, landen.

1.6.3 Einige Angriffsszenarien

... oder »Was passieren kann, wenn keiner aufpasst«. OWASP nennt in der Dokumentation der Top 10 [OWASP_Top10-2017] beispielhaft folgende drei Angriffsszenarien:

Beispiel 1: unerkannter Angreifer löscht Code-Repository und Foren-Inhalte

Der Server einer von einem kleinen Team entwickelte Open-Source-Forensoftware wurde über eine Schwachstelle in der eigenen Software kompromittiert. Der Angreifer konnte das interne Sourcecode-Repository mit der nächsten Version der Software sowie alle Foren-Inhalte löschen. Obwohl der Sourcecode wiederhergestellt werden konnte, führt der Mangel

an Monitoring, Logging oder Alarmierung zu einem viel schlimmeren Leck. Infolgedessen ist das Projekt nicht mehr aktiv.

Ich finde dieses Beispiel gar nicht mal so schlecht, nur: Warum hat der Angreifer das Sourcecode-Repository gelöscht? Außer er handelt im Auftrag einer Konkurrenz-Software oder ist einfach nur an Vandalismus interessiert, hat er doch gar nichts davon. Viel besser (für ihn) wäre es doch gewesen, unbemerkt eine Hintertür in die nächste Version der Forensoftware einzubauen, über die er dann später alle Server, die diese Software betreiben, kompromittieren kann. Da ja wohl nichts überwacht wurde, wäre so eine Manipulation sehr wahrscheinlich unbemerkt geblieben.

Beispiel 2: Brute-Force-Angriff, ganz langsam und gemütlich

Ein klassischer Brute-Force-Angriff besteht darin, für den gewünschten Benutzernamen in kurzer Zeit alle möglichen Passwörter auszuprobieren. Das ist zum einen sehr auffällig, wird zum anderen im Allgemeinen durch einen Brute-Force-Schutz verhindert.

Wenn dem Angreifer egal ist, welches Benutzerkonto er übernimmt, und er es auch nicht eilig hat, kann er einen viel weniger auffälligen Weg wählen und nach irgendeinem Benutzer mit einem üblichen Passwort suchen, um dann dieses Konto zu übernehmen. Er geht dann einfach eine Liste mit Benutzernamen durch und probiert für jeden das gleiche Passwort aus. Für alle nicht betroffenen Benutzer erzeugt dieser Scan lediglich einen einzigen fehlgeschlagenen Login-Versuch. Nach einigen Tagen kann er den Angriff mit einem anderen Passwort ausprobieren usw. Ohne ausreichendes Monitoring bleibt dieser Angriff unentdeckt.

Beispiel 3: Virenscanner-Warnungen werden ignoriert

Ein großer Einzelhändler in den USA nutzte eine interne Analyse-Sandbox zur Prüfung von Anhängen auf Schadsoftware. Sie erkannte zwar potentiell unerwünschte Software, es reagierte aber niemand auf diesen Bericht. Die Sandbox hatte schon seit einiger Zeit Warnungen ausgegeben, bevor einer externen Bank aufgrund betrügerischer Kartentransaktionen auffiel, dass die Rechner des Einzelhändlers kompromittiert waren.

Dass Warnungen ignoriert werden, ist ein häufiger Fehler. Aber da scheint mir auch eine Fehlkonfiguration vorzuliegen: Wieso wurden die verdächtigen Anhänge an die Benutzer ausgeliefert, so dass sie Schaden anrichten konnten? Eigentlich müssten sie doch in Quarantäne genommen oder sogar gelöscht werden, aber auf keinen Fall dürften sie die Empfänger erreichen.

1.6.4 Unzureichendes Logging und Monitoring verhindern

In allen drei Fällen hätte der Angriff durch gutes Logging und Monitoring sehr wahrscheinlich komplett abgewehrt, zumindest aber frühzeitig gestoppt werden können. Je nach Schutzbedarf der gespeicherten bzw. verarbeiteten Daten sind dafür verschiedene Maßnahmen nötig:

- Allgemein gilt: Stellen Sie sicher, dass jeder Login sowie jeder von der Zugriffskontrolle und der serverseitigen Eingabenüberprüfung erkannte Fehler protokolliert wird, und zwar mit so viel Benutzerkontext, wie nötig ist, um verdächtige oder bösartige Benutzerkonten zu identifizieren. Und das für einen ausreichend langen Zeitraum, damit die Daten auch noch für eine spätere forensische Analyse zur Verfügung stehen.
  Dabei sind allerdings im Allgemeinen Datenschutzvorschriften zu beachten – Sie können nicht einfach anfangen, alle Daten für alle Ewigkeit zu speichern. Personenbezogene Daten, zu denen außer der Benutzerkennung auch die verwendete IP-Adresse gehört, dürfen nur im Rahmen der gesetzlichen Vorschriften gespeichert werden.
- Stellen Sie sicher, dass alle erzeugten Logfiles in einem Format erzeugt werden, das von einer zentralisierten Logfile-Management-Lösung verarbeitet werden kann.
- Stellen Sie sicher, dass für hochwertige Transaktionen ein Audit Trail mit Integritätskontrollen angelegt wird, um Manipulationen oder Löschungen zu verhindern. Dies kann z. B. durch die Nutzung von »Append-only«-Datenbanktabellen erreicht werden.
- Führen Sie eine effektive Überwachung und Alarmierung ein, so dass verdächtige Aktivitäten frühzeitig erkannt und danach angemessen darauf reagiert werden kann.
- Führen Sie einen Incident-Response-und-Recovery-Plan ein, um im Falle eines Angriffs auf vorher festgelegte Verfahren zurückgreifen zu können.

Für den Schutz der Anwendungen gibt es kommerzielle und Open-Source-Lösungen, beispielsweise den *OWASP AppSensor* [OWASP_AppSensor], der eine Intrusion Detection auf Anwendungsbasis bietet, oder die Web Application Firewall *ModSecurity* [ModSecurity], die z. B. mit dem *OWASP ModSecurity Core Rule Set* [OWASP_ModSecRules] konfiguriert werden kann. Diese Tools erzeugen (bei richtiger Konfiguration) ausreichende Logfiles, die Sie dann natürlich auch auswerten müssen, wofür es spezielle Log-Correlation-Software gibt, die oft mit anpassbaren Dashboards und verschiedenen Alarmfunktionen ausgestattet ist. Das bekannteste Beispiel für so eine Software ist wohl *Nagios* [Nagios].

1.7 Links

- [BeEF] BeEF – The Browser Exploitation Framework Project
  http://beefproject.com
- [CostBreach] »Ponemon Institute's 2017 Cost of Data Breach Study: Global Overview«
  https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=SEL03130WWEN
- [Kali_Linux] Kali Linux
  https://www.kali.org
- [Mini_MySqlat0r] Mini MySqlat0r
  https://web.archive.org/web/20130313113739/http://www.scrt.ch/en/attack/downloads/mini-mysqlat0r

- [ModSecurity] ModSecurity
  https://modsecurity.org
- [Nagios] Nagios
  https://www.nagios.com
- [Nikto] Nikto
  https://cirt.net/nikto2
- [Nmap] Nmap
  https://nmap.org
- [OWASP_AppSensor] OWASP AppSensor
  https://www.owasp.org/index.php/OWASP_AppSensor_Project
- [OWASP_ModSecRules] »OWASP ModSecurity Core Rule Set (CRS)«
  https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project
- [OWASP_Top10] »OWASP Top Ten Project«
  https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- [OWASP_Top10-2017] »OWASP Top 10 – 2017 – The Ten Most Critical Web Application Security Risks«
  https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
- [OWASP_Top10-2017-RC2] »OWASP Top 10 – 2017 – The Ten Most Critical Web Application Security Risks – Release Candidate 2«
  https://www.owasp.org/images/b/b0/OWASP_Top_10_2017_RC2_Final.pdf
- [OWASP_ZAP] OWASP Zed Attack Proxy Project
  https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
- [skipfish] skipfish
  https://code.google.com/archive/p/skipfish/ und *https://tools.kali.org/web-applications/skipfish*
- [THC-Hydra] THC-Hydra
  https://tools.kali.org/password-attacks/hydra
- [Turnkey-VM] TurnKey LAMP-Stack
  https://www.turnkeylinux.org/lamp
- [w3af] w3af
  http://w3af.org
- [Wapiti] Wapiti
  https://sourceforge.net/projects/wapiti/
- [Zenmap] Zenmap
  https://nmap.org/zenmap/

Kapitel 8

Dateioperationen: von Directory-Traversal bis Datei-Uploads

In diesem Kapitel dreht sich zuerst alles um Schwachstellen, die den Zugriff auf normalerweise nicht zugängliche Dateien erlauben: Directory-Traversal, die damit verwandte Local File Inclusion und deren großen, bösen Bruder Remote File Inclusion. Und weil sie thematisch hier am besten passten, werden auch Datei-Uploads kurz behandelt.

Jede Webanwendung besteht aus mehr oder weniger vielen verschiedenen statischen oder dynamisch erzeugten Seiten, die von einem Webserver an den Benutzer ausgeliefert werden. Schauen wir uns gemeinsam an, welche Probleme dabei entstehen können.

8.1 Directory-Traversal

Egal ob statisch oder dynamisch erzeugt: Seiten sind im Wesentlichen Dateien auf dem Server, die vom Webserver und von der Webanwendung beim Aufruf gegebenenfalls mit dynamischen Inhalten erweitert und dann ausgeliefert werden.

Da nicht alle auf dem Server gespeicherten Dateien für den Aufruf durch den Benutzer gedacht sind, ist der Webserver auch dafür zuständig, nur die für die Benutzer vorgesehenen Dateien auszuliefern und Zugriffe auf alle anderen Dateien zu verhindern.

Dazu werden die Dateien im Wesentlichen in zwei Kategorien unterteilt: zum einen die für die Ausgabe an den Benutzer bestimmten Dateien, die in einem eigenen Verzeichnis und seinen Unterverzeichnissen gespeichert sind, zum anderen alle anderen Dateien, die außerhalb dieses sogenannten *Webroot-Verzeichnisses* gespeichert sind. Ein Spezialfall sind dabei Dateien, die zwar unterhalb des Webroot-Verzeichnisses gespeichert sind, auf die aber nur bestimmte Benutzer zugreifen dürfen. Die Verzeichnisse könnten z. B. so aussehen wie in Listing 8.1 und sind so auch als Zugabe in der Demo-Anwendung enthalten, so dass Sie damit bei Bedarf selbst in der Shell experimentieren können.

```
/etc/
   passwd
   web/
      .htpasswd
```

```
/var/
   www/
      index.html
      admin/
         .htaccess
         index.html
      berichte/
         .htaccess
         index.html
      daten/
         index.html
      privat/
         .htaccess
         .htpasswd
         index.html
      webanwendung/
         .htaccess
         index.html
```

Listing 8.1 Ein Beispiel für ein Verzeichnis

Im Verzeichnis `/etc` liegt zum einen die Datei `passwd`, die früher unter Unix und damit auch Linux die Passwörter der Benutzer enthielt. Inzwischen enthält sie nur noch die Liste der Benutzer, die Passwörter sind in einer versteckten, geschützten Datei gespeichert. Als Beispiel für Zugriffe auf eine Datei mit vertraulichen Informationen dient `/etc/passwd` aber nach wie vor.

Außerdem gibt es hier das Verzeichnis `web/` mit Konfigurationsdateien für den Webserver, hier insbesondere die Datei `.htpasswd` mit den Benutzern des Webservers und ihren Passwörtern. Im Gegensatz zu `/etc/passwd` sind die Daten in den `htpasswd`-Dateien (nicht nur hier, sondern gegebenenfalls auch an anderen Orten) durchaus von Relevanz.

Im Verzeichnis `/var` gibt es das Unterverzeichnis `www/`, das das Webroot-Verzeichnis darstellt. Zugriffe auf den Webserver werden in dieses Verzeichnis geleitet. Beim Aufruf von z. B.

```
http://www.server.example/index.html
```

wird vom Webserver die Datei

```
/var/www/index.html
```

ausgeliefert.

Das Webroot-Verzeichnis enthält im Beispiel die folgenden Unterverzeichnisse. Über die in einigen Verzeichnissen vorhandene Datei `.htaccess` wird der Apache-Webserver speziell für

dieses Verzeichnis konfiguriert, z. B. um einen Verzeichnisschutz einzurichten oder die Ausgabe eines Verzeichnislistings abweichend von der Defaultkonfiguration ein- oder auszuschalten.

- `admin/` mit dem Administrationsbereich, auf den nur einige Benutzer zugreifen dürfen, deren Passwörter in der Datei `/etc/web/.htpasswd` festgelegt sind.
- `berichte/` enthält Geschäftsberichte. Es gibt keine per Default angezeigte Datei wie z. B. die `index.html`-Datei, und die Ausgabe eines Verzeichnislistings ist in der `.htaccess` für dieses Verzeichnis ausgeschaltet. Außerdem gibt es keinen Zugriffsschutz. Jeder, der einen gültigen Dateinamen kennt, kann diese Datei aufrufen.
- `daten/` enthält die Daten für die Webanwendung, z. B. die auszugebenden Informationen wie die Texte von Impressum, Datenschutzerklärung, Info-Seiten ...
- `privat/` ist ein Verzeichnis mit vertraulichen Informationen, auf die nur die Benutzer zugreifen dürfen, die in der lokalen `.htaccess`- und `.htpasswd`-Datei festgelegt sind.
- `webanwendung/` enthält die Skripte der Webanwendung und alles, was sonst noch dafür benötigt wird.

8.1.1 Verzeichnis wechsle dich

Als *Directory-Traversal* wird der Wechsel von einem Verzeichnis in ein anderes bezeichnet. In der Shell ebenso wie in Webanwendungen wird die Zeichenkette `../` bzw. `..\` verwendet, um in das über dem aktuellen Verzeichnis liegende Verzeichnis zu wechseln.

Das funktioniert sogar in der URL: Wenn im Beispiel oben statt

```
http://www.server.example/index.html
```

die URL

```
http://www.server.example/anwendung/../index.html
```

aufgerufen wird, wird trotzdem der Inhalt von

```
http://www.server.example/index.html
```

ausgegeben, da `/anwendung/../` das Gleiche wie `/` ist: Erst wird in das Verzeichnis `/anwendung` gesprungen, danach über die `../` zurück in das darüberliegende Verzeichnis `/`. Das funktioniert oft sogar mit einem Verzeichnisnamen, den es gar nicht gibt, z. B.

```
http://www.server.example/gibtsnicht/../index.html
```

Da der Pfad vor dem Zugriff auf das Dateisystem in eine einheitliche Form gebracht wird (was als *Kanonisierung* bezeichnet wird), hebt das `../` das nicht vorhandene Verzeichnis `gibtsnicht/` auf, bevor es beim Zugriff darauf einen Fehler auslösen kann.

So besonders hilfreich ist das natürlich nicht, denn man kann diesen Pfad ja auch direkt aufrufen. Interessant ist ein Directory-Traversal-Angriff nur, wenn darüber der Zugriff auf sonst nicht zugängliche Dateien, z. B. */etc/passwd* oder eine *.htpasswd*-Datei, möglich ist. Deren Aufruf über ihre URL verhindert der Webserver nämlich (sofern er richtig konfiguriert ist).

Für einen Directory-Traversal-Angriff muss der Angreifer also einen Parameter manipulieren, über den

1. auf eine Datei zugegriffen wird und der
2. nicht (ausreichend) geprüft oder gefiltert wird, so dass die Manipulation erfolgreich ist.

Je nach Art der Datei, auf die zugegriffen wird, und der Art des Zugriffs können die Folgen von der Preisgabe vertraulicher Informationen (Zugriff auf z. B. eine Datei mit Zugangsdaten mit anschließender Ausgabe) bis zur Ausführung unerwünschten Codes (Zugriff auf ein Programm oder eine Skriptdatei mit anschließender Ausführung) reichen.

Ein Beispiel

Für die folgenden Beispiele soll sich die Webanwendung im Verzeichnis `webanwendung/` im Webroot-Verzeichnis befinden, Daten wie auszugebende Texte befinden sich im Verzeichnis `daten/` im Webroot-Verzeichnis. Ein normaler Benutzer ruft beispielsweise

```
http://www.server.example/webanwendung/index.php
```

auf, um die Webanwendung zu starten. Das Skript `index.php` besitzt den Parameter `zeige`, über den vor der Ausgabe einzufügende Daten festgelegt werden können. Nehmen wir an, der Benutzer lässt sich die Datenschutzerklärung anzeigen. Nach dem Klick auf den entsprechenden Link ändert sich die URL in

```
http://www.server.example/webanwendung/index.php?zeige=datenschutz
```

und die Datenschutzerklärung wird ausgegeben.

Ich gehe davon aus, dass wir wissen, dass die Daten sich im Verzeichnis `daten/` befinden. Das macht das Folgende leichter zu erklären. Wie man den Speicherort herausbekommt, erkläre ich später noch. Die Datei mit der Datenschutzerklärung müsste sich also unter

```
http://www.server.example/daten/datenschutz
```

befinden. Beim Aufruf dieses Links wird der Text der Datenschutzerklärung ausgegeben, aber unformatiert und ohne das Aussehen der anderen Seiten der Anwendung. Das Skript `index.php` wandelt also (Text-)Dateien in optisch an die Anwendung angepasste Webseiten um. Ob das Skript weitere Funktionen hat und wenn ja, welche, ist zurzeit uninteressant.

Was passiert, wenn als Wert für den Parameter `zeige` eine andere Datei als die im Rahmen der Anwendung üblichen angegeben wird?

Ein Standardbeispiel für solche Fälle ist für Windows-Systeme die Datei *boot.ini* und wie oben schon erwähnt für Unix-artige Systeme die Datei */etc/passwd*. Mit dem Aufruf

```
http://www.server.example/webanwendung/index.php?zeige=/etc/passwd
```

ist der Angriff nicht erfolgreich. Er führt zum Laden der Datei

```
/var/www/daten//etc/passwd
```

und die gibt es ja nicht. Sehr wahrscheinlich wird der doppelte Slash darin auch einen Fehler auslösen. Um auf die Datei */etc/passwd* zugreifen zu können, ist ein Directory-Traversal nötig, der Wechsel des Verzeichnisses: Durch Eingabe von `../` wird in das übergeordnete Verzeichnis gewechselt. Der Einfachheit halber wird hier immer der Slash / als Pfadtrenner verwendet, Windows-Systeme verwenden gegebenenfalls den Backslash \.

Am Pfad sieht man nicht, wie viele Verzeichniswechsel nötig sind. Wir wissen zwar, dass die Datenschutzerklärung sich in der Datei

```
/daten/datenschutz
```

befindet, aber der erste / ist in diesem Pfad das Webroot-Verzeichnis und nicht das Wurzelverzeichnis des Dateisystems. Mit dem Aufruf

```
http://www.server.example/webanwendung/index.php?zeige=../etc/passwd
```

kommen wir also nicht ans Ziel. Zum Glück kann man aber mit den Directory-Traversal-Sequenzen nicht über das /-Verzeichnis hinaus wechseln, ein paar `../` zu viel stören also nicht. Der Aufruf von

```
http://www.server.example/webanwendung/index.php?zeige=../../../../../etc/passwd
```

führt daher zur Ausgabe des Inhalts von `/etc/passwd`, jedenfalls sofern der Webserver auf diese Datei zugreifen darf. Ist ihm der Zugriff nicht erlaubt, wird auch nichts ausgegeben. Aber das macht ja nichts, darin steht ja sowieso nichts Interessantes. Viel interessanter sind die `.htaccess`- und `.htpasswd`-Dateien, und auf die darf der Webserver natürlich zugreifen. Der Aufruf von

```
http://www.server.example/webanwendung/index.php?zeige=../webanwendung/.htaccess
```

gibt die `.htaccess`-Datei im Verzeichnis `webanwendung/` aus.

Das funktioniert natürlich nur, wenn der Wert des Parameters auch tatsächlich den Dateinamen festlegt. Wird z. B. noch eine Endung wie `.txt` oder `.html` angehängt, können auch nur Dateien dieses Typs aufgerufen werden. Sofern nicht über einen Trick der angehängte Teil ausgeblendet werden kann, aber dazu komme ich noch weiter unten.

Aber vielleicht gibt es ja noch mehr zu entdecken.

Hochzählen kann hilfreich sein

An dieser Stelle möchte ich noch kurz auf einen Angriff hinweisen, der nichts mit Directory-Traversal zu tun hat, aber an dieser Stelle relativ gut passt.

Aus der Erkundung der Webanwendung kennen wir alle gültigen Werte für den Parameter `zeige`. Eventuell sind darunter Werte, die sich hochzählen lassen, beispielsweise numerische IDs, bekannte Produktbezeichnungen oder Datumsangaben, so dass aus dem Vorhandensein von z. B. `bericht2016`, `bericht2017` und `bericht2018` auf einen noch nicht verlinkten `bericht2019` geschlossen werden kann. Auch dadurch ist der Zugriff auf (noch) nicht für die Veröffentlichung bestimmte Informationen möglich.

Das Gleiche gilt sinngemäß für direkte Zugriffe: Wenn wir wissen, dass im Verzeichnis `berichte/` z. B. die Dateien `jahresabschluss16.pdf`, `jahresabschluss17.pdf` und `jahresabschluss18.pdf` verlinkt sind, lohnt sich immer auch der Versuch, auf die Datei `jahresabschluss19.pdf` zuzugreifen, auch wenn sie noch nirgends verlinkt ist. Vielleicht liegt sie aber schon auf den Server.

8.1.2 Quelltext ausgeben lassen

Aber zurück zum Directory-Traversal. Zur Erinnerung: Bisher hatte ich erst einmal angenommen, dass der Angreifer bereits weiß, aus welchem Verzeichnis die über den Parameter `zeige` angeforderten Dateien geladen werden, denn mit diesen Informationen lässt sich der Directory-Traversal am einfachsten erklären. Ab jetzt gehen wir davon aus, dass der Angreifer diese Information (noch) nicht hat.

Um weiterzukommen, fehlen ihm also Informationen über den Aufbau der Webanwendung. Um an sie zu gelangen, könnte er zuerst einmal versuchen, folgende URL aufzurufen:

```
http://www.server.example/webanwendung/index.php?zeige=index.php
```

Der Webserver führt die Anweisungen in `index.php` aus und gibt dabei den Inhalt der über den Parameter `zeige` spezifizierten Datei aus: In diesem Fall ist das der Quelltext der Skriptdatei. Mit etwas Glück (für den Angreifer, für den Betreiber der Webanwendung ist das natürlich Pech) wird dabei der Code nicht ausgeführt, sondern als Klartext ausgegeben.

Nach dem gleichen Muster kann ein Angreifer sich dann auch alle anderen Skriptdateien anzeigen lassen. Sie enthalten meist viele für den Angreifer nützliche Informationen wie z. B. die Zugangsdaten zu einer Datenbank, Verweise auf nachgeladene Konfigurationsdateien oder Informationen über die Programmlogik. Im Beispiel könnte der Angreifer so beispielsweise herausfinden, aus welchem Verzeichnis die nachgeladenen Dateien geholt werden.

8.1.3 Datei nicht gefunden

Eventuell wird der Quelltext aber auch nicht ausgegeben, sondern es erscheint eine Fehlermeldung, dass die Datei nicht gefunden wurde, oder der Platz für die auszugebenden Daten

bleibt leer. Da die Datei eindeutig vorhanden ist, lässt das darauf schließen, dass der Wert des Parameters nicht direkt für den Zugriff auf die Datei verwendet, sondern zuvor um weitere Daten ergänzt wird. Dafür gibt es zwei übliche Ansätze, die auch gemeinsam auftreten können: Es wird ein fest vorgegebener Pfad und/oder eine fest vorgegebene Endung hinzugefügt.

8.1.4 Ein fester Pfad als Präfix schreit nach Directory-Traversal

Ein fester Pfad ist für den Directory-Traversal-Angriff kein Problem, ganz im Gegenteil: Beim Directory-Traversal geht es ja gerade darum, aus dem vorgegebenen Pfad auszubrechen. Der Angreifer muss also nur noch herausfinden, wo auf dem Server er sich befindet und wohin er muss. Das ist bei ausführlichen Fehlermeldungen oft sehr leicht, wird ohne jede weiterführende Information nur durch reines Raten dagegen ziemlich schwierig, vor allem, wenn noch eine feste Endung angehängt wird wie z. B. `.php`, `.html` oder `.txt`.

8.1.5 Eine feste Endung als Suffix ist ungünstiger

Eine eventuell angehängte Endung kann etwa durch das Anhängen eines NULL-Bytes (`%00`) an den Wert des Parameters umgangen werden: Das NULL-Byte wird als Ende des Strings mit Pfad und Dateinamen interpretiert, eine möglicherweise noch folgende, fest kodierte Endung folglich ignoriert. Klappt das nicht, ist nur der Zugriff auf Dateien mit dieser festen Endung möglich, was je nach Endung aber auch schon gefährlich werden kann.

Jetzt gehen wir einmal davon aus, dass sich im Verzeichnis `webanwendung/` nur die Skripte der Webanwendung befinden und alle davon verwendeten Texte im Verzeichnis `daten/` liegen. Im Beispiel kann das z. B. so aussehen:

```
http://www.server.example/webanwendung/index.php?zeige=../index.php%00
```

ergibt wieder die `"Datei nicht gefunden"`-Meldung oder eine leere Ausgabe, ebenso wie

```
http://www.server.example/webanwendung/index.php?zeige=../../index.php%00
```

und weitere derartige Versuche. Aber mit

```
http://www.server.example/webanwendung/index.php?zeige=../webanwendung/index.php%00
```

erreicht der Angreifer sein Ziel: Der Quelltext der Skriptdatei wird ausgegeben. Darin steht für dieses Beispiel, dass für den Zugriff auf die anzuzeigenden Dateien der Pfad

```
/daten/
```

vor den Wert des Parameters `zeige` kopiert wird. Davon ausgehend, kann der Angreifer nun versuchen, weitere für ihn interessante Dateien ausgeben zu lassen, etwa die oben erwähnten `boot.ini` oder `/etc/passwd`:

```
http://www.server.example/webanwendung/index.php?zeige=../../../../boot.ini
http://www.server.example/webanwendung/index.php?zeige=../../../../etc/passwd
```

8.1.6 Ausführliche Fehlermeldungen erfreuen den Angreifer

Die mühsame Suche nach der Skriptdatei bleibt dem Angreifer erspart, wenn die Fehlermeldung nicht nur einfach "`Datei nicht gefunden`" oder ähnlich lautet, sondern ausführlicher ist. Eine ausführliche Fehlermeldung für den Aufruf von

```
http://www.server.example/webanwendung/index.php?zeige=index.php
```

könnte z. B. so aussehen:

```
Warnung: Die Datei /var/www/webanwendung/daten/index.php.txt konnte nicht gelesen werden!
```

Danach weiß der Angreifer, wo nach der anzuzeigenden Datei gesucht wird und dass `.txt` als feste Endung angehängt wird. Er kann also sofort die passende Anzahl `../`-Sequenzen und Verzeichnisnamen für den Wechsel in ein gewünschtes Verzeichnis einfügen und über `%00` die störende Endung ausklammern.

8.1.7 Directory-Traversal ohne Kenntnis des Startverzeichnisses

Auch wenn der Angreifer nicht weiß, in welchem Verzeichnis er startet, ist er nicht hilflos: Die Webseiten werden je nach verwendeten Webserver in bestimmten Standardverzeichnissen gespeichert, von denen nur sehr selten abgewichen wird. Mit welchem Webserver und welcher Version er es zu tun hat, hat der Angreifer schon beim Sammeln der Informationen über die Webanwendung ermittelt. Er kann nun ausgehend von diesen Standardverzeichnissen und dem angezeigten Pfad versuchen, eine bekannte Datei aufzurufen. Dabei kommt ihm zugute, dass die meisten Dateisysteme redundante Directory-Traversal-Sequenzen, die über die Wurzel des Dateisystems hinausführen würden, tolerieren. Es darf also ruhig mehr als die benötigte Anzahl Directory-Traversal-Sequenzen eingegeben werden – solange sie mindestens bis zur Wurzel des Dateisystem reichen, wird der Pfad danach ab der Wurzel neu begonnen.

8.1.8 Freie Auswahl

Hat der Angreifer einen Parameter gefunden, durch dessen Manipulation er sich frei auf dem Server bewegen kann, kann er danach beliebige Dateien betrachten. Für die Nutzung der dabei gesammelten Informationen gibt es im Wesentlichen zwei Möglichkeiten: Die Informationen werden für weitergehende Angriffe (z. B. ausgespähte Zugangsdaten, Erkenntnisse über die Programme) oder direkt verwendet (Verkauf von ausgespähten Betriebsgeheimnissen oder Kreditkartendaten ...).

8.1.9 Schreiben ist besser als Lesen

Bisher habe ich nur Angriffe mit lesenden Zugriffen beschrieben. Directory-Traversal-Angriffe sind aber auch über Parameter möglich, die eine zu schreibende Datei definieren. Für einen Angreifer ist das natürlich ein Glücksfall, da er darüber beliebige Befehle auf dem

Server ausführen kann. Dazu kann er beispielsweise ein Skript im Startup-Verzeichnis eines Benutzers erzeugen, Konfigurationsdateien ändern oder Skripte in ein Verzeichnis schreiben, aus dem er sie starten kann.

8.1.10 To write or not to write?

Während bei einem Lesezugriff ein erfolgreicher Angriff sofort an der Ausgabe der aufgerufenen Datei zu erkennen ist, ist das Gleiche bei einem Schreibzugriff sehr viel schwieriger. Ein möglicher Test besteht darin, sowohl eine für alle Benutzer beschreibbare Datei als auch eine nicht einmal für den Administrator oder *root* beschreibbare Datei anzugeben. Für Windows könnten das z. B.

```
../../../../../../../../../../schreibtest
```

und

```
../../../../../../../../../../windows/system32/config/sam
```

sein, für Unix-artige Systeme

```
../../../../../../../../../../tmp/schreibtest
```

und

```
../../../../../../../../../../tmp
```

Der Versuch, ein Verzeichnis mit einer Datei zu überschreiben, sollte immer zu einer Fehlermeldung führen.

Eine weitere Testmöglichkeit ist das Schreiben in eine neue Datei unterhalb des Webroot-Verzeichnisses, die danach im Webbrowser aufgerufen wird. Das setzt aber voraus, dass dem Angreifer die Lage des Webroot-Verzeichnisses bekannt ist und der Benutzer, mit dessen Rechten die Datei geschrieben wird, dort Schreibberechtigung hat.

Je nachdem, was und wo geschrieben werden kann, reichen die Folgen so eines Angriffs vom Denial of Service (Überschreiben wichtiger Dateien) über das Einschleusen von falschen Informationen oder Code zum Verbreiten von Drive-by-Infektionen bis zum Umgehen der Authentifizierung und zum Einschleusen von Code.