

- StudentDocuments -
Eine Webanwendung mit Sicherheitslücken

Sebastian Kölle, Stefanie Reinicke, Emilia Wittmers

Seminar Secure Web Application Engineering
Wintersemester 2008/09
Hasso-Plattner-Institut, Universität Potsdam

Zusammenfassung

Entwickeln von Webanwendungen ist oft nicht bewusst, welche Angriffsmöglichkeiten bestimmte Sicherheitslücken bieten. Wenn sie selbst in die Rolle des Angreifers schlüpfen, können sie dies nachvollziehen und beim Entwickeln berücksichtigen. Die Internet-Plattform Tele-Lab bietet dafür einen legalen Rahmen. In Lektionen und Übungen können dort Angriffe in einer abgeschlossenen Umgebung erprobt werden.

Im Seminar Secure Web Application Engineering wurde ein Übungsszenario für das Tele-Lab entwickelt. Die dabei entstandene Webanwendung „StudentDocuments“ ermöglicht das Speichern und Herunterladen von Dokumenten verschiedener Nutzer. Absichtlich eingebaute Sicherheitslücken lassen sich unter anderem mittels SQL-Injection und einer Webshell ausnutzen. Die in PHP entwickelte Webanwendung wurde in einer Virtuellen Maschine mit Ubuntu-Server eingerichtet. Das Übungsszenario beinhaltet zudem drei konkrete Aufgabenstellungen mit Musterlösung.

1 Einleitung

Kontext Die internetbasierte Lernplattform *Tele-Lab* bietet Schülern, Studenten und anderen Interessierten eine Übungsumgebung, mit der sie Angriffe auf Systeme simulieren und nachvollziehen können, um das eigene Sicherheitsbewusstsein zu schärfen. Das am Lehrstuhl *Internet Technologien und Systeme* des Hasso-Plattner-Instituts entwickelte System stellt für diesen Zweck eine Tutoring-Software für web-basiertes Training, sowie virtuelle Maschinen für die Ausführung praktischer Übungen bereit [WM08]. Im Seminar *Secure Web Application Engineering* haben wir unter der Überschrift *Insecure Web Application Engineering* ein Übungsszenario entwickelt, welches wir in diesem Dokument vorstellen.

Konzept Zentraler Bestandteil des Szenarios ist die Webanwendung *StudentDocuments*, die beabsichtigte Sicherheitslücken aufweist. Diese sind vom Nutzer mittels einer Anleitung, die in einer Lerneinheit vorgestellt wird, aufzuspüren. Zu den Themen gehören SQL-Injections, Angriffe auf Authentifizierungsmechanismen, der Einsatz von Webshells und das Knacken von Passwörtern, auf die im nächsten Abschnitt kurz eingegangen wird¹.

Anschließend stellen wir die Funktionen der Webanwendung und die Übungsszenarien vor. Im Abschnitt Musterlösung wird Schritt für Schritt der Angriffsvorgang geschildert. Abschließend beschreiben wir die technische Realisierung und erläutern, wie die Sicherheitslücken eingebaut wurden. Im Anhang findet sich außerdem eine detaillierte Anleitung für die Installation und Konfiguration des Webservers.

¹Eine ausführliche Beschreibung der Angriffstechniken befindet sich in [Web07]

2 Hintergrund: Zu verwendende Angriffstechniken

SQL-Injection SQL (Structured Query Language) ist eine deklarative Anfragesprache für relationale Datenbanken. Bei SQL-Injection handelt es sich um eine spezielle Art der Code-Injection. Werden auf Webseiten Eingaben der Nutzer direkt in Datenbankabfragen eingefügt, lassen sich mittels SQL-Injection zusätzliche, vom Angreifer beabsichtigte Anfragen ausführen.

Angriffe auf Authentifizierungsmechanismen Um sich bei einer Webanwendung zu authentifizieren, wird vom Nutzer häufig die Eingabe seiner Zugangsinformationen gefordert. Ist es dem Angreifer möglich, Nutzernamen und Passwörter eines anderen Nutzers herauszufinden, kann er sich mit fremder Identität anmelden. Zur Überprüfung vergleicht die Webanwendung dabei in der Regel nicht die Klartext-Passwörter, sondern zwei Werte, die mittels einer Einwegfunktion berechnet wurden.

HTTP ist ein zustandsloses Protokoll. Webseiten mit Nutzerauthentifizierung nutzen deshalb beispielsweise Cookies, um feststellen zu können, welcher Nutzer gerade eine Seite angefordert hat. HTTP-Cookies sind Name-Wert-Paare, die auf dem Client gespeichert und beim nächsten Besuch der Seite zum Server gesendet werden.

Wenn es einem Angreifer gelingt, sich einen Cookie zu generieren, der einem anderen Nutzer „gehört“, kann er sich beispielsweise als jemand anderes ausgeben und Sicherheitsmechanismen wie Passwort-Authentifizierung umgehen.

Webshell Eine Webshell ist ein Programm, das es dem Angreifer ermöglicht, über den Browser beliebige Kommandozeilen-Befehle auf dem Webserver auszuführen. Gelingt es dem Angreifer, z.B. eine PHP-Webshell auf einem Server hochzuladen, kann er darüber unter Umständen den ganzen Webserver kontrollieren.

3 Das Szenario

3.1 Die Anwendung: StudentDocuments

StudentDocuments ist eine Webanwendung, mit der Studenten Dokumente, wie z.B. Vorlesungsmitschriften, Referate oder Hausarbeiten, ihren Kommilitonen zur Verfügung stellen können.

Die Anwendung aus Nutzersicht Bevor ein Nutzer die Dokumentenverwaltung verwenden kann, muss er sich zunächst einmal registrieren. Zur Registrierung wird ein Login-Name, seine E-Mail-Adresse und ein Passwort benötigt. Mit den eingegebenen Daten kann er sich anschließend bei der Anwendung anmelden und alle Funktionen nutzen.

Unter der Rubrik *Mein Profil* ist es dem Nutzer möglich seine Daten einzusehen und zu ändern. Seine hochgeladenen Dokumente findet er unter *Meine Dokumente*. Dort kann er weitere Dokumente hinzufügen und diese gegebenenfalls auch wieder löschen. Mithilfe der Funktion *Suchen* kann der Nutzer die Dokumente seiner Kommilitonen finden und sich diese herunterladen. Eine Vorschau der Dokumente ermöglicht, das Herunterladen von nicht gewünschten Dateien zu vermeiden.

Sicherheitsvorkehrungen Die Webanwendung nutzt unter anderem Nutzerauthentifizierung und Überprüfung von Dateitypen als Sicherheitsmaßnahmen:

- Kein Nutzer kann die Daten anderer Nutzer einsehen.
- Bei der obligatorischen Registrierung ist es erforderlich, ein mindestens 6 Zeichen langes Passwort zu wählen.
- Einem normalen Nutzer ist es nicht möglich, die Dokumente eines fremden Nutzers zu löschen.
- Bei der Funktion zum Hochladen von Dokumenten gibt es eine Überprüfung des Dateityps. Es ist nur erlaubt, Dateien der Typen jpg, png, txt und pdf hochzuladen.



Abbildung 1: Die Suchfunktion in der Webanwendung 'StudentDocuments'

- Der Administrator der Webanwendung (ein Nutzer mit Admin-Rechten) kann über ein gesondertes Admin-Interface im Gegensatz zu normalen Nutzern auch fremde Dokumente löschen.

Mögliche Angriffsstellen Trotz der Sicherheitsvorkehrungen ist es Nutzern möglich, durch gezielte Angriffe auf *StudentDocuments*, beispielsweise die persönlichen Daten der anderen Nutzern einzusehen oder auf deren hochgeladene Dokumente zuzugreifen, um diese zu löschen. Eine Sicherheitslücke in der Suchfunktion ermöglicht das Auslesen der Nutzerdaten mittels einer SQL-Injection. Auch die Manipulation von Cookies kann dazu verwendet werden, sich als Admin anzumelden. Ein Zugriff auf die Datenbank ist dadurch möglich, dass die `.htaccess`-Datei im Browser anzeigbar ist. Durch eine unsaubere Implementierung der Upload-Funktion kann ein Angreifer eine Webshell hochladen und diese für seine Zwecke ausnutzen.

3.2 Die Aufgabenstellung

Das Szenario bietet drei Aufgaben mit steigendem Schwierigkeitsgrad. Dadurch soll eine differenzierte Bewertung der Lerneinheit möglich sein.

Einfache Aufgabe Der Angreifer soll sich mit einer falschen Identität anmelden. Dazu muss er das Passwort eines bestimmten, bereits vorhandenen Nutzers - nennen wir ihn *Bob* - knacken. Die Aufgabe gilt als gelöst, wenn er als *Bob* ein Text-Dokument mit Bobs Passwort im Klartext hochgeladen hat.

Schwere Aufgabe Ziel dieser Aufgabe ist der Zugriff auf die Datenbank der Webanwendung über phpMyAdmin. Als Beweis für den Zugriff soll der Angreifer das Passwort des Admins in der Datenbank ändern - dies ist nicht über die Weboberfläche durchführbar.

Bei der Lösung ist es notwendig, sich die Identität eines Nutzers mit Admin-Rechten zu beschaffen. Da dies nicht wie in der einfachen Aufgabe durch Knacken des Passwortes möglich ist (das Passwort ist komplexer gewählt), muss der Angreifer eine andere Lösung finden.

Bonus-Aufgabe Zum Abschluss soll der Angreifer die vollständige Kontrolle über den Webserver erlangen. Dazu soll er eine Webshell, die ihm in der eigenen virtuellen Maschine zur Verfügung gestellt ist, hochladen und ausführen lassen. Mit ihrer Hilfe kann er nun bestimmte Dateien auf dem Webserver löschen oder manipulieren. Diese Aufgabe gilt als erfolgreich abgeschlossen, wenn bestimmte Server-Logdateien gelöscht wurden.

3.3 Welche Kenntnisse werden vorausgesetzt?

Einfache Aufgabe

- SQL (speziell MySQL) und SQL-Injection
- Knacken von Passwort-Hashes, speziell MD5

Schwere Aufgabe

- Grundkenntnisse in HTML
- grundlegendes Verständnis des HTTP-Protokolls, speziell Cookies
- Browserfunktionen (z.B. im Firefox) zur Cookie-Manipulation
- Base64-Codierung
- Absicherung einer Website mit htaccess (Apache)
- Pfadangaben im Dateisystem
- Tools zum Knacken von Passwörtern (z.B. John the Ripper)

Bonus-Aufgabe Für die Bonus-Aufgabe genügt es, zusätzlich mit einer gegebenen Webshell umgehen zu können, sowie Geduld und Kreativität bei der Lösungsfindung mitzubringen.

4 Musterlösung

Im Folgenden wird eine Komplettlösung der Aufgaben Schritt für Schritt beschrieben. Mit „Angreifer“ ist die Person gemeint, die die Aufgabe bearbeitet.

4.1 Einfache Aufgabe

Anwendung ausspähen Um sich einen Überblick über die Anwendung und mögliche Sicherheitslücken zu verschaffen, registriert sich der Angreifer zunächst. Lässt er dabei beispielsweise das Feld für die E-Mail-Adresse aus, erhält er die Fehlermeldung (A), die ihn auf den Namen einer Tabelle hinweist.

Nachdem er sich erfolgreich registriert und angemeldet hat, untersucht er die nun sichtbaren Seiten. Die Dokumentensuche liefert bei leerem Suchfeld eine Liste von Dokumenten mit dem dazugehörigen Nutzer. Die Eingabe eines einfachen Hochkommata (1) liefert einen Hinweis (B), dass sich das Eingabefeld für eine SQL-Injection eignet. Hängt der Angreifer eine Raute ans Ende des Ausdrucks (2), gibt es keine Fehlermeldung, da die störenden Anführungszeichen im Rest der Zeile auskommentiert wurden.

Eingaben im Suchfeld

- (1) '
- (2) ' #
- (3) ' UNION SELECT * FROM users #
- (4) ' UNION SELECT 1,2,3,4,5,6,7,8 from users #
- (5) ' UNION SELECT 1,id,3,4,5,6,password,id from users #

Fehlermeldungen

- (A) Es müssen alle Felder ausgefüllt werden. Der Datensatz konnte nicht in der Users-Tabelle gespeichert werden.
- (B) You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 3
- (C) The used SELECT statements have a different number of columns.

SQL-Injection Nun heißt es, weiter zu probieren - Wie lassen sich mithilfe der Dokumentensuche Informationen aus der Users-Tabelle anzeigen? Ein Ansatzpunkt ist der SQL-Befehl UNION, der mehrere Anfragen aneinanderhängt, vorausgesetzt die Spaltenanzahl stimmt überein. Die Eingabe des Befehls (3) liefert die Fehlermeldung (C). Das bedeutet also, die Nutzertabelle heißt tatsächlich *users*, aber die Anzahl der Spalten stimmt noch nicht überein.

Um herausfinden, wie viele Spalten die eigentliche SQL-Anfrage benutzt, wählt der Angreifer Konstanten im SELECT-Teil. Erfolg hat er mit dem Ausdruck (4), mit dem die Anwendung unterhalb der Dokumente die Werte 2 und 7, sowie den Login-Namen des Nutzers mit der Id 8 anzeigt.

Da beim Registrierungsformular unter anderem *Login-Name* und *Passwort* abgefragt wurden, setzt der Angreifer diese Begriffe an die entsprechenden Stellen der SQL-Injection. Vermutlich heißen die Spalten in der Datenbanktabelle genauso. Zudem ist es wahrscheinlich, dass die Users-Tabelle eine Spalte *id* enthält. Der Befehl (5) lässt die Anwendung eine Liste mit Ids, Passwort-Hash und dem Nutzernamen anzeigen. Somit ist der schwierigste Teil der einfachen Aufgabe geschafft.

Passwort knacken Da die Passwort-Hashes jeweils genau 32-Zeichen lang sind und nur Zeichen von 0-9 sowie a-e enthalten, liegt die Vermutung nahe, dass es sich dabei um MD5²-Hashes handelt.

Da Bob leider ein einfaches Passwort gewählt hat, lässt es sich schnell herausfinden. Kurze Passwörter, die aus bekannten Begriffen (z.B. Vornamen) bestehen, lassen sich auch auf Online-Seiten in Sekunden knacken.

Da der Angreifer nun Bobs Login-Daten kennt, lädt er zum Abschluss der Aufgabe eine Textdatei mit dem geknackten Passwort in Bobs Dokumentensammlung.

4.2 Schwere Aufgabe

Cookie-Manipulation Die SQL-Injection der einfachen Aufgabe lieferte eine Liste mit ID, Login-Namen und Passworthashes. Einer der Nutzer („admin“) hat vermutlich Administrator-Rechte und Zugriff auf die Datenbank. Leider kann der Angreifer nicht wie bei der einfachen Aufgabe das Passwort knacken - es ist kryptisch und lang gewählt.

Durch Cookie-Manipulation ist es jedoch möglich, sich ohne Kenntnis des Passworts als „Admin“ anzumelden. Mittels des Firefox-Addons „Add N Edit Cookies“ können Cookies ausgelesen und verändert werden. Dazu meldet sich der Angreifer wie bei der vorigen Aufgabe an und liest das bei ihm hinterlegte Cookie aus. Dieses sieht beispielsweise folgendermaßen aus:

```
Altes Cookie
name:      user
content:   aWQ9NTQ7a2V5PTIwMDkwMjIwMTgONjQ0XzU0MjMONQ%3D%3D
```

Bei %3D handelt es sich eigentlich um =, die Zeichen wurden für die Übertragung mittels HTTP ersetzt. Da der Cookie offensichtlich Buchstaben des Standard-Alphabets und Ziffern enthält, könnte es sich um eine Base64-Codierung handeln. Typisch für Base64 sind außerdem bis zu zwei Gleichheitszeichen am Ende. Das Ergebnis der Dekodierung³, für die vorher alle %3D durch = ersetzt wurden, ist folgendes:

```
id=54;key=20090220184644_542345
```

Da „Admin“ in der Datenbank mit der ID „1“ gespeichert war, ersetzt der Angreifer die Zahl entsprechend und nimmt die Base64-Codierung vor. Anschließend müssen gegebenenfalls die Gleichheitszeichen wieder ersetzt werden.

```
Neues Cookie
name:      user
content:   aWQ9MTtrZXk9MjAwOTAyMjAxODQ2NDRfNTQyMzQ1
```

²Der Message Digest Algorithmus 5 ist eine verbreitete Hashfunktion, die einen 128-Bit langen Hashwert erzeugt. Dieser wird oft als 32 Zeichen lange Hexadezimalzahl dargestellt. MD5-Passwort-Hashes lassen sich zwar nicht direkt entschlüsseln, jedoch kann der Angreifer solange potentielle Passwörter MD5-verschlüsseln, bis der Hash mit dem ursprünglichen Passwort-Hash übereinstimmt.

³Base64-Decoder und Encoder gibt es auch als Online-Version.

Wird nun das Cookie auf den neuen Wert gesetzt, kann der Angreifer auf der Willkommenseite den Link zum Admin-Bereich benutzen. Dort kann er sich unter der Nutzerverwaltung Adminrechte geben, um beim nächsten Anmelden weniger Aufwand zu haben. Im Menüpunkt „phpMyAdmin“ findet sich ein Link zur Datenbank-Administration. Doch bevor der Angreifer direkten Zugriff auf die Datenbank hat, muss er Login-Name und Passwort herausfinden.

Anzeige der htaccess-Datei Ebenfalls auf der phpMyAdmin-Seite (1) befindet sich ein Hinweis auf eine *Hypertext-Zugriffsdatei*, kurz `.htaccess`. Nun zahlt sich besondere Aufmerksamkeit aus. Betrachtet der Angreifer den Seitenquelltext der Dokumentenvorschau einer beliebigen Datei (hier Eingebettetes System.pdf, URL (2)), findet er den Pfad für ein eingebettetes Bild (3). Setzt er den Aufruf von `view.php` mit dem *file*-Parameter anstelle der `preview.php` in die URL (4), wird die Datei direkt im Browser angezeigt.

1) `http://localhost/phpmyadmin/index.php`

2) `http://localhost/preview.php?file=f1d5cefb215966a15e1521b47b1312cb`

```
3) <div id="preview">

```

...

4) `http://localhost/view.php?file=f1d5cefb215966a15e1521b47b1312cbEingebette.png`

Löscht der Angreifer nun ein paar Zeichen des *file*-Parameters, sieht er eine Seite mit folgender Warnung, die verrät, dass Dateien aus dem Verzeichnis `documents/preview` direkt angezeigt werden.

Warning: file_get_contents(documents/preview/f1d5cefb2966a15e1521b47b1312c...

Nun versucht der Angreifer, den *file*-Parameter zu manipulieren, um vielleicht die `.htaccess`-Datei anzeigen zu können. Dabei zeigt (5) ebenso wie (4) die Vorschau der Datei an. (6) öffnet dagegen die PDF-Datei, die sich im Ordner `documents` befindet, wobei die Dateiendung nicht im Namen gespeichert ist.

5) `http://localhost/view.php?file=../preview/f1d5cefb215966a15e1521b47b1312cbEingebette.png`

6) `http://localhost/view.php?file=../f1d5cefb215966a15e1521b47b1312cbEingebette`

7) `http://localhost/view.php?file=../../phpmyadmin/index.php`

8) `http://localhost/view.php?file=../../phpmyadmin/.htaccess`

9) `http://localhost/view.php?file=../../phpmyadmin/.htpasswd`

Da der Angreifer den Link zu phpMyAdmin kennt (1), versucht er auf eine andere Weise auf die Startseite zuzugreifen (7). Er sieht nun den Quelltext der `index.php` und weiß, dass er sich im richtigen Verzeichnis befindet. Ruft er (8) auf, sieht er den Inhalt der `.htaccess`-Datei:

```
AuthType Basic
AuthName "Adminbereich"
AuthUserFile /var/www/phpmyadmin/.htpasswd
AuthGroupFile /dev/null
require valid-user
```

Diese enthält einen Verweis auf eine `.htpasswd`-Datei, die das für die Authentifizierung verwendete Passwort enthält. Sie lässt sich nun auch leicht über (9) anzeigen:

```
admin:f1.pjr3bXG7XQ
```

Der Benutzername ist nun klar, jedoch handelt es sich bei `f1.pjr3bXG7XQ` keineswegs um das Klartext-Passwort.

Passwort knacken Bevor der Angreifer nun das Passwort des Admins in der Datenbank ändern kann, muss er das Passwort der `.htpasswd`-Datei entschlüsseln. Durch ein Programm zum Knacken von Passwörtern (z.B. John the Ripper) dürfte das jedoch kein Problem darstellen. Das Original-Passwort (`admin5`) ist sehr leicht gewählt. Beim Ändern des Admin-Passworts in der Tabelle `users` zur Erfüllung der Aufgabe ist noch die MD5-Verschlüsselung zu beachten.

4.3 Bonus-Aufgabe

Das erste Ziel dieser Aufgabe ist es, eine Webshell auf dem Webserver zu plazieren und Zugriff darauf zu erlangen. Die Webshell liegt dem Angreifer in Form einer einzelnen php-Datei vor, die durch die Dokumenten-Upload Funktion der Anwendung auf dem Server gespeichert werden muss.

Der Angreifer versucht zuerst, die Webshell direkt mit der Upload-Funktion hochzuladen. Dabei erhält er jedoch die Fehlermeldung, dass nur PDF-, Bilddateien (Jpeg, PNG) und reine Textdateien erlaubt sind. Er muss also zunächst die Upload-Funktionalität der Anwendung genauer untersuchen, um Rückschlüsse auf das Verhalten zu gewinnen. Dazu werden am besten Dokumente verschiedener Dateitypen hochgeladen.

Wie schon bei der schweren Aufgabe liefert der Quellcode der Dokumentenvorschau den entscheidenden Hinweis:

```

```

Wie aus der schweren Aufgabe bekannt ist, gibt der `file`-Parameter an, welche Datei im Verzeichnis `documents/preview` als Vorschau geladen werden soll. Experimente mit verschiedenen Dateien zeigen, dass der Name eines hochgeladenen Dokumentes aus 32 hexadezimalen Ziffern (offensichtlich ein Hashwert) und zehn weiteren Zeichen besteht. Die zehn Zeichen sind dabei der Anfang des Dateinamens, wobei Leerzeichen durch Unterstriche ersetzt wurden. Mit dem Wissen, dass die Dateien aus dem Verzeichnis `documents/preview` geladen werden, kann der Angreifer nun direkt auf die Vorschaudateien zugreifen:

```
http://localhost/documents/preview/2745c5088c981dd75f4f5ebeaa205a8bProtokoll_.png
```

Allerdings handelt es sich hierbei nur um die Vorschau, für die Textdateien unter anderem gekürzt werden und somit die Webshell unbrauchbar machen. Um die Originaldatei zu laden, muss der Angreifer jedoch lediglich auf den gleichen Dateinamen ohne Dateieindung `.png` im Verzeichnis `documents` zugreifen⁴:

```
http://localhost/documents/2745c5088c981dd75f4f5ebeaa205a8bProtokoll_
```

Nun muss der Angreifer noch die Webshell so auf den Webserver laden, dass er sie ausführen kann. Dazu muss er den Dateinamen der Webshell modifizieren, um das Abschneiden des Dateinamens beim Upload auszunutzen. Dies geschieht, indem er die Endung `.txt` an den Namen hängt und gleichzeitig so kürzt oder verlängert, dass er genau 14 Zeichen lang ist:

```
webshell.php => webshe.php.txt
```

Beim Upload wird der Dateiname auf zehn Zeichen gekürzt, womit der Dateinhalt als `<HASH>webshe.php` im Verzeichnis `documents` gespeichert wird. Auf dieses kann der Angreifer nun per Webbrowser zugreifen⁵:

```
http://localhost/documents/0123456789abcdeffedcba9876543210webshe.php
```

Mit der Webshell hat der Angreifer nun Zugriff auf den Server mit den Rechten des Webserver (`www-data`). Will er nun beispielsweise die Logs unter `/var/log` löschen oder manipulieren, muss er noch `root`-Rechte erlangen. Dies ist auf dem gegebenen Server jedoch sehr einfach, da das Programm `sudo` falsch konfiguriert wurde. Es erlaubt dem Benutzer `www-data` ohne Passwortheingabe Befehle mit `root`-Rechten auszuführen:

```
sudo rm -r /var/log/*
```

⁴Um auf diese Idee zu kommen reicht das Anschauen von `http://localhost:8080/documents`, da hier alle Dateien aufgelistet werden

⁵der für den Dateinamen notwendige Hashwert lässt sich über die Dokumentenvorschau ermitteln

5 Technische Realisierung

Im folgenden soll ein kurzer Überblick über die Architektur der Software gegeben werden. Weiterhin wird beschrieben, wo im Programm welche Sicherheitslücken vorgesehen wurden, um die Angriffe aus den Aufgaben zu ermöglichen.

Architektur Die Architektur gliedert sich in drei Schichten. Die unterste Schicht (*Models*) kommuniziert mit der Datenbank und stellt Methoden zur Verfügung um User- und Document-Objekte aus der Datenbank zu lesen und zu schreiben. Außerhalb dieser Schicht findet keine direkte Kommunikation mit der Datenbank statt.

Die mittlere Schicht (*Services*) stellt Dienste bereit, die den größten Teil der Anwendungslogik implementieren. Unterschieden werden dabei folgende drei Dienst-Typen:

Authorization-Dienste zum Beispiel das Registrieren eines neuen Nutzers, Anmeldung und Abmeldung

Dokumenten-Dienste zum Beispiel das Hochladen eines Dokuments oder das Löschen von Dokumenten

Benutzer-Dienste zum Beispiel das Verändern der Rechte eines Nutzers

Die oberste Schicht (*Presentation*) ist schließlich für die Verarbeitung der Nutzereingaben und für die Darstellung der Anwendung verantwortlich. Für die reine Darstellung wird die Template-Engine *Smarty* [Sm09] verwendet, die die Ergebnisse der Anwendung in HTML-Templates darstellt.

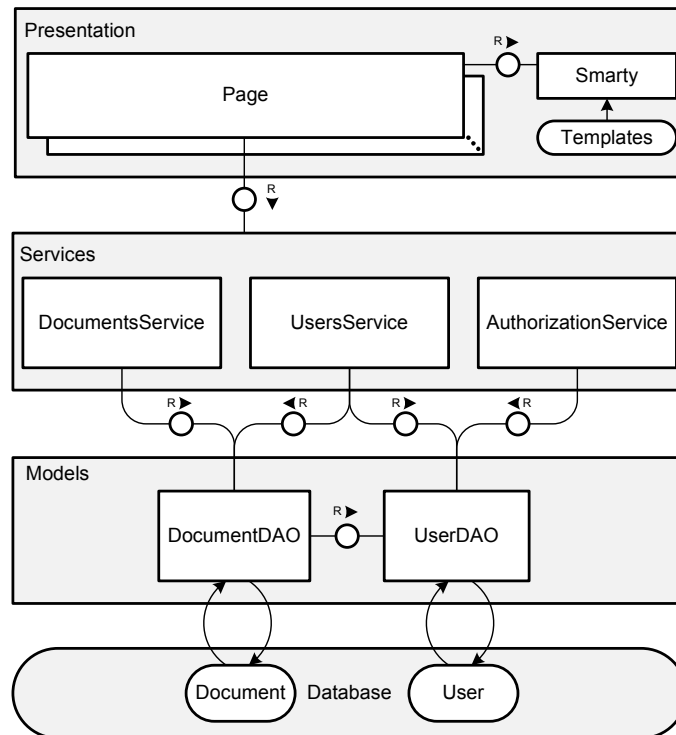


Abbildung 2: Überblick über die Architektur

Sicherheitslücken der einfachen Aufgabe Der wesentliche Teil der einfachen Aufgabe ist die SQL-Injection. Diese wird durch eine Sicherheitslücke in der Models-Schicht ermöglicht, in der die von außen kommenden Eingaben (z. B. Suchbegriffe) direkt in die SQL-Anfragen an die Datenbank eingebaut werden. Eine Filterung der Eingaben findet auch in den darüber liegenden Schichten nicht statt.

Um die Sicherheitslücke ausnutzbar zu machen, war es weiterhin notwendig, die PHP-Option `magic_quotes_gpc` auf auszuschalten. Anderenfalls fügt PHP in Eingaben selbstständig Escape-Zeichen für gängige Sonderzeichen hinzu, die überlicherweise für SQL-Injections verwendet werden. Dies würde die die Ausnutzung der Sicherheitslücke erheblich erschweren.

Sicherheitslücken der schweren Aufgabe Zur Lösung der schweren Aufgabe müssen im Wesentlichen zwei Sicherheitslücken ausgenutzt werden: Zum einen muss der Angreifer sein Session-Cookie manipulieren, um Administrator-Rechte zu erlangen. Die dafür verantwortliche Sicherheitslücke liegt im AuthorizationService. Der dort eingesetzte Algorithmus zur Bestimmung des Session-Tokens ist so schwach, dass der Angreifer durch Manipulation seines eigenen Tokens zu jedem beliebigen Nutzer werden kann. Die Implementierung dieser Sicherheitslücke erforderte den Verzicht auf den php-eigenen Session-Mechanismus und die Implementierung eines eigenen, schwachen Algorithmus.

Weiterhin benötigt der Benutzer für diese Aufgabe die Möglichkeit, sich beliebige Dateien auf dem Server anzusehen. Die dazu notwendige Sicherheitslücke befindet sich in der Präsentationsschicht (`view.php`). Diese benutzt den übergebenen Parameter direkt als Name für die zu öffnende Datei, statt diesen anhand einer ID aus der Datenbank abzufragen.

Sicherheitslücken der Bonusaufgabe Die Bonusaufgabe wird im Wesentlichen durch zwei Schwachstellen ermöglicht: Erstens, die Dokumenten- und Vorschauverzeichnisse sind über den Webbrowser erreichbar, was das Ausführen darin enthaltender Dateien ermöglicht. Zweitens, die künstliche Begrenzung des Dateinamens auf zehn Zeichen nach dem Hash erlaubt es dem Angreifer, die für die Umgehung des Filters notwendige Endung `.txt` abzuschneiden. Die Datei kann damit als reguläre PHP-Datei ausgeführt werden.

6 Zusammenfassung

In diesem Dokument wurde eine Webanwendung für das Tele-Lab vorgestellt, die bewusst mit Sicherheitslücken versehen wurde. Der Fokus der Anwendung lag auf einer Plattform, über die Studenten Dokumente online veröffentlichen können. Benutzern ermöglicht diese Anwendung, eigenständig Sicherheitslücken zu finden und auszunutzen.

Es wurden drei Übungsaufgaben entwickelt, bei denen jeweils eine oder mehrere der Schwachstellen ausgenutzt werden müssen, um ein bestimmtes Ziel zu erreichen. Zu diesen Aufgaben wurden die entsprechenden Musterlösungen dokumentiert und erläutert.

Der Anwender soll mit Hilfe von StudentDocuments die Möglichkeit haben, sich praktisch mit der Sicherheit von Webanwendungen auseinanderzusetzen. Er bekommt die Chance, selbst einmal in Rolle des Angreifers zu schlüpfen und erhält einen Eindruck davon, welche Sicherheitslücken in Webanwendungen auftreten können und welche Folgen diese haben.

Auch für uns als Entwickler hat die Arbeit an dieser Anwendung interessante Erkenntnisse gebracht: Zum einen ist es nicht trivial, eine Webanwendung mit glaubhaften Sicherheitslücken zu entwickeln, die sich gleichzeitig leicht ausnutzen lassen. Zum anderen kann ein guter Programmierstil die Wahrscheinlichkeit von Sicherheitslücken erheblich verringern, da Fehlerquellen ausgeschlossen werden.

Literatur

- [Sm09] Smarty Template Engine. <http://www.smarty.net/>.
- [SmI09] Smarty Installation Guide. http://www.smarty.net/quick_start.php.
- [Web07] Dafydd Stuttard und Marcus Pinto. *The web application hacker's handbook: discovering and exploiting security flaws*. John Wiley & Sons, Inc., New York, NY, USA, 2007.
- [WM08] Christian Willems und Christoph Meinel. Awareness Creation mit Tele-Lab IT-Security: Praktisches Sicherheitstraining im virtuellen Labor am Beispiel trojanischer Pferde. In *Sicherheit*, Seiten 513–532, 2008.

A Installationsanleitung

Im Folgenden wird beschrieben, wie die virtuelle Maschine für studentDocuments eingerichtet wurde. Informationen über verwendete Softwareversionen sind in Tabelle 1 zu finden, gewählte Benutzernamen und Kennwörter in Tabelle 2.

Anlegen der virtuellen Maschine mit VirtualBox Zunächst soll eine virtuelle Maschine angelegt werden. In VirtualBox wurden dazu folgende Einstellungen verwendet:

Name studentDocuments

Typ Ubuntu

Hauptspeicher 256MB

Festplatte 8GB, dynamisch wachsend

Sonstiges PAE/NX aktiviert

Ubuntu Server Installation In diesem Schritt wird Ubuntu Server 8.10 installiert. Folgende Einstellungen wurden verwendet:

Sprache deutsch, deutsches Tastaturlayout

Partitionierung Standardvorschlag:

- Partition 1 mit ext3
- Partition 5 mit Swap

Benutzer siehe Tabelle 2

Automatische Updates deaktiviert

Software LAMP server

Installation zusätzlicher Pakete aus den Repositories Folgende Pakete werden zusätzlich zur Standardinstallation benötigt. `build-essential` und `linux-headers-2.6.27-7-server` sind notwendig, um die `VBoxLinuxAdditions` (siehe unten) zu installieren.

- build-essential
- linux-headers-2.6.27-7-server
- imagemagick

Installation der VBoxLinuxAdditions Um die Arbeit mit der virtuellen Maschine zu erleichtern, wurden die `VBoxLinuxAdditions` installiert. Mit diesen lassen sich zum Beispiel Shared Folders definieren, die zum Datenaustausch zwischen Host und Guest dienen.

Software	Version
Ubuntu Server	8.10
Apache	2.2.9-7ubuntu3
PHP	5.2.6-2ubuntu4
MySQL	5.0.67-0ubuntu6
Imagemagick	7:5.3.7.9.dfsg1-2ubuntu3
phpMyAdmin	3.1.1
Smarty	2.6.22
Sun xVM Virtual Box	2.0.4

Tabelle 1: Verwendete Versionen

	Benutzer	Kennwort
Server	student	insecure09
Datenbank	root	Hza7wz5r1_ui
phpMyAdmin (HTTP-Auth)	admin	admin5
Administrator	admin	A1d2M3i4N5

Tabelle 2: Zugangsdaten

Weitere Konfiguration Die folgenden Optionen wurden geändert, um die Sicherheitslücken der Anwendung zu ermöglichen:

htaccess aktivieren unter `/etc/apache2/sites-available/default` für das Verzeichnis `/var/www/` die Option `AllowOverride All` setzen

Magic Quotes von PHP deaktivieren unter `/etc/php5/apache2/php.ini` Option `magic_quotes_gpc = Off` setzen

www-data das Ausführen von allen Programmen ohne Passwortabfrage erlauben mit `sudo visudo` folgende Zeile ergänzen: `www-data ALL = NOPASSWD: ALL`

Installation von Smarty Zur Darstellung der Templates muss Smarty installiert sein. Folgende Schritte sind notwendig⁶

- Kopieren des Inhalts des `libs`-Verzeichnisses nach `/usr/share/php`
- Anlegen von `/var/smarty/templates_c`, Schreibrechte für Benutzer `www-data`

Einrichten der Port-Weiterleitung von VirtualBox Um auf Apache auf der virtuellen Maschine zugreifen zu können, muss in VirtualBox die entsprechende Port-Weiterleitung aktiviert werden:

- `VBoxManage setextradata „studentDocuments“ „VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestwww/Protocol“ TCP`
- `VBoxManage setextradata „studentDocuments“ „VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestwww/GuestPort“ 80`
- `VBoxManage setextradata „studentDocuments“ „VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestwww/HostPort“ 8080`

Installation der Anwendung Nun können die Skripte und Resourcen der Anwendung in den Ordner `/var/www` entpackt und der Datenbank-Dump geladen werden:

- Entpacken der Datei `studentdocuments.tar.gz` nach `/var/www`
- Entpacken der Datei `documents.tar.gz` nach `/var/www/documents`
- Anlegen einer Datenbank `student_documents` und Import der Daten aus `dbdump.sql.gz`
- gegebenenfalls Anpassen der Konfigurationsdatei `config.php`

Datenbank füllen Alternativ zum Datenbank-Dump kann das Ruby-Skript `DataCreator.rb` genutzt werden. Dieses erstellt 50 zufällige Nutzer und lädt Dateien aus einem Verzeichnis hoch. Allerdings muss der Admin-Nutzer manuell angelegt werden.

⁶Detaillierte Informationen finden sich in der Smarty-Dokumentation[SmI09]