Webapplication-Security

Version 1.0

Inhaltsverzeichnis

Für wen ist dieses Dokument gedacht?

Dieses Dokument ist meines Wissens eines der grössten und umfangreichsten zum Thema Webapplication-Security, das in deutscher Sprache frei im Internet existiert. Somit ist es für Webprogrammierer und Sicherheitsverantwortliche interessant, gleichzeitig kann man es aber auch als Whitepaper in diesem Bereich ansehen. Ich hoffe auf eine grosse Auswahl an Personen, die Interesse an den hier besprochenen und demonstrierten Angriffen und Verteidigungsarten haben und werde mich deswegen auch nicht auf eine bestimmte Zielgruppe konzentrieren.

Einführung

Als Erstes möchte ich kurz erläutern, was man unter dem Begriff “Webapplication-Security“ versteht und was für Themen ich im folgenden Teil dieses Dokumentes genauer besprechen werde.

Webapplication-Security kann man mit Webapplikations-Sicherheit eins zu eins ins Deutsche übersetzen. Somit ist es nicht notwendig, noch genauer um was es geht.

Nun wollen wir uns der Frage widmen, welche Arten von Sicherheitsproblemen, betreffend Webapplikationen, es überhaupt gibt. Es existieren zum Beispiel Einbrüche in den Webserver, Cross-Site Scripting und SQL-Injection, und natürlich noch viele mehr. Im Folgenden werde ich auch weniger verbreitete Angriffsmethoden anschneiden und Wichtigere im Detail erklären.

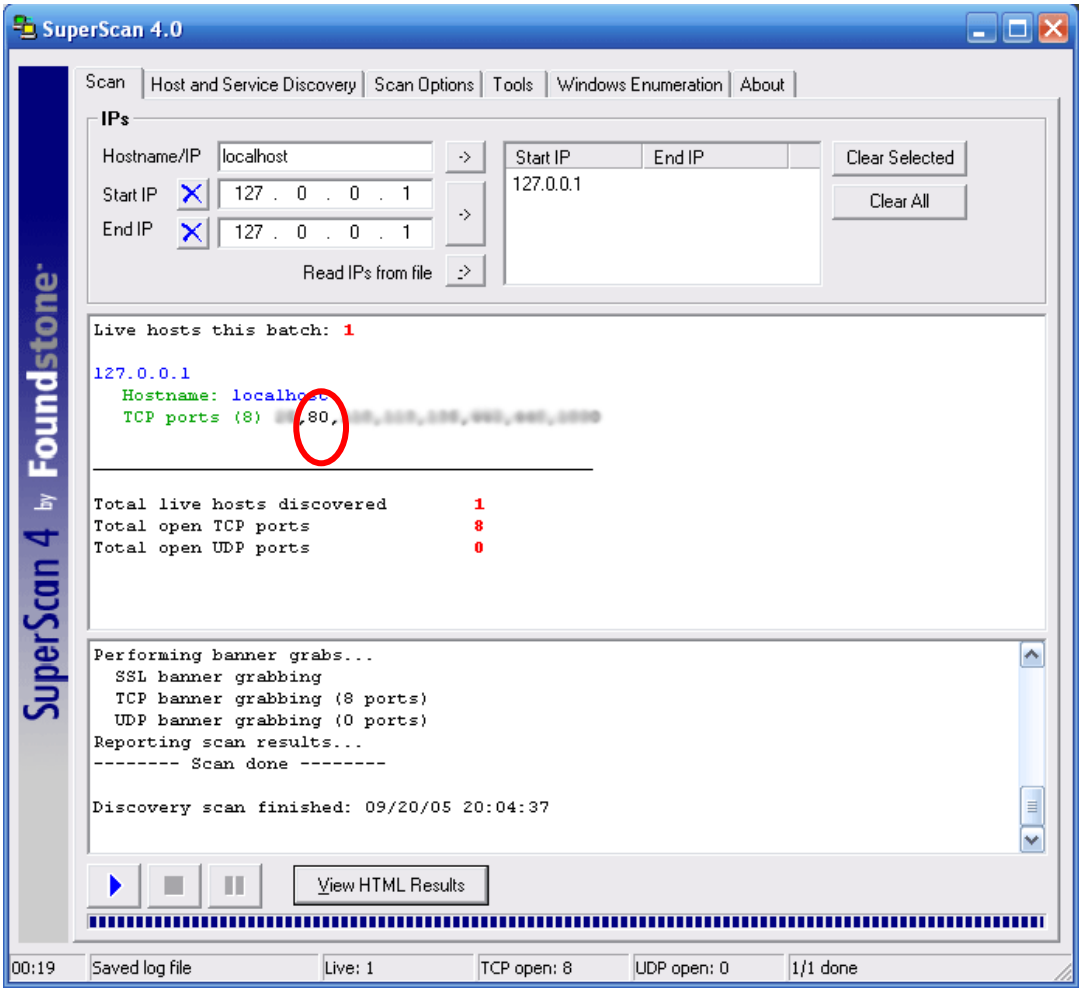
Einbruch in den Webserver

Das erste Szenario für Angriffe im Bereich Webapplication-Security ist nicht ein eigentlicher Teil davon, es geht um den Angriff auf einen Webserver. Dieser erfolgt weder über die Webapplikation, noch ist er selbst eine.

Hierzu näher auf einzelne Angriffe einzugehen wäre nicht sonderlich sinnvoll, denn diese sind von Server zu Server unterschiedlich (je nach OS, Version, Applikationen, welche benutzt werden). Jedoch möchte ich kurz auf die am häufigsten benützte Webserver-Software eingehen: den Apache-Webserver.

Den Apache-Webserver direkt anzugreifen, bedarf einer guten Vorbereitung und Analyse des Systems, denn auch hier gilt wie bei jedem Angriff auf eine Applikation: „Keine Version ist gleich zu behandeln wie eine Andere.“ Als Erstes lohnt es sich somit herauszufinden, welche Apache-Version auf dem Server eingesetzt wird. Um dies festzustellen zu können, wird meist ein Portscanner eingesetzt, ich habe mich in diesem Fall mit SuperScan 4.0 ausgerüstet.

Um das weitere Vorgehen besser zu verstehen, führen wir nun einen Scan des Systems durch, auf welchem der Apache-Server läuft. Die könnte etwa wie folgt aussehen:



Wie Sie in dem roten Kreis sehen, ist nach getaner “Arbeit“, dass der Port 80 auf welchem das http läuft tatsächlich offen ist, wie er es auch sollte.

Nun gehen wir weiter und schauen uns die genaueren Daten des Scan’s an:

TCP Port	Banner
80 World Wide Web HTTP	HTTP/1.1 200 OK Date: Tue, 20 Sep 2005 18:04:29 GMT Server: Apache/2.0.54 (Win32) mod_ssl/2.0.54 OpenSSL/0.9.7g PHP/5.0.4 Last-Modified: Sun, 06 Feb 2005 22:00:16 GMT ETag: "102fa-be-d737bc00" Accept-Ranges: bytes Content-Length: 190 Connection: close Content-Type: text/html; charset=ISO-8859-1

1 2

Wir finden nun alle wichtigen Infos, die wir für einen Angriffsversuch auf das System benötigen.

1. Im ersten Teil erkennt man die installierte Version des Apache-Servers, in diesem Falle *2.0.54*. Weiter können wir (in der Klammer) sehen, dass es sich um ein Windows-System handelt.
2. Hier sehen wir auch gleich noch, welche Version von PHP eingesetzt wird. Daraus können wir ableiten, wie das System wahrscheinlich konfiguriert ist, bzw. wir kennen somit die Standardkonfigurationen.

Nun wissen wir schon eine ganze Menge mehr über das besagte System, doch nun kommt der eigentliche Teil des Angriffs, nämlich das gezielte Ausnutzen von Schwachstellen.
Meist wird dies über so genannte Exploits durchgeführt. Dies sind kleine Programme, die automatisch eine Sicherheitslücke ausnützen. Die Kunst besteht grösstenteils nur darin, den passenden Exploit zu finden. Hierfür kann man sich diversen verfügbaren Datenbanken bedienen, zum Beispiel jener von www.securityfocus.com.

Darauf wie Exploits eingesetzt werden, möchte ich hier nicht näher eingehen, sondern nur sagen, dass, wer sich damit beschäftigen möchte, die Programmiersprachen C sowie Assembler beherrschen sollte/muss und es darüber auch genügend freie Literatur gibt.

An dieser Stelle möchte ich auch darauf hinweisen, dass dies wirklich nur eine einzelne Möglichkeit für den Angriff eines Webservers war, tatsächlich gibt es jedoch tausende davon. Es ist zum Beispiel nicht zwingend notwendig, den Server nur auf Port 80 zu scannen und die darüber kommunizierenden Applikationen (im Beispiel den Apache-Server) anzugreifen, denn es gibt unzählige andere Applikationen, die eine Verbindung nach Aussen haben und für die es Exploits gibt. Doch all dies werden Sie selbst schnell erkennen, wenn Sie sich näher mit Exploiting beschäftigen.

Input-Validation Angriffe

Gehen wir nun auf den wohl häufigsten begangenen Fehler von Web-Programmierern ein: nicht sauber überprüfte Eingaben. Bei Beispielen werde ich mich auf PHP beschränken, da diese Scriptsprache am meisten verbreitet ist, jedoch lässt sich das Ganze auch z.B. auf ASP.net übertragen.

Vertrauen ist gut, Kontrolle ist besser. Diesen Satz haben wir sicher alle schon zur Genüge gehört, doch sollte man sich spätestens als Programmierer einmal überlegen, in wie fern da auch etwas dran ist.

Am besten werde ich dies an einem kleinen Beispiel zeigen:

```
<?php
      echo $variable;
?>
```

Wenn wir diesen Code haben, interessiert es uns nicht, was die Variable zum Beispiel für einen Datentyp hat. Somit könnte ein User mit http://www.domain.com/index.php?variable=Hallo den Text „Hallo“ ausgeben. Was ist denn das Problem dabei? Damit könnte man unter Umständen schon das ein oder andere Cross-Site Scripting vornehmen (siehe nächstes Kapitel).

Als kleinen Einschub möchte ich hier auch noch einen schwerwiegenden Programmierfehler, der leider immer noch öfters gemacht wird, vorstellen. Es handelt sich dabei um einen Input-Validation Angriff auf die PHP-Funktion include().
Auch hier möchte ich das Problem anhand eines Beispiel-Scripts erläutern.

```
<?php
      include($variable);
?>
```

Was wir hier sehen, ist ein ganz normaler include-Befehl, wie man ihn in fast jeder Webapplikation findet. Der Unterschied liegt hier jedoch darin, dass die einzubindende Datei Variabel ist. Dadurch, dass hier keine Prüfung der Variable vorgenommen wird, könnte man jegliche Datei in dieses Script einbinden. Dies wäre eigentlich schon genug schlimm, jedoch können wir dadurch sogar auf eigentlich nicht für das Web freigegebene Dateien zugreifen und diese Einbinden. Ein Angreifer könnte zum Beispiel Folgendes für $variable einsetzen:

../../../../../../../etc/passwd

Damit würde er die Datei mit den verschlüsselten Passwörtern bekommen.

Doch möchte ich nochmals ein etwas praxisnäheres Beispiel zeigen um Ihnen auch klar zu machen, dass solche kleinen Fehler schnell in einem Desaster enden können.

```
<?php
if (angemeldeter_user())
            {
            $zugang = true;
            }
if ($zugang)
            {
            Include “/secret/geheim.php“;
            }
?>
```

Wir sehen, durch die Funktion “angemeldeter_user“ wird geprüft ob der Benutzer angemeldet ist oder nicht. Ist er angemeldet, erhält er Zugang zur Datei “/secret/geheim.php“, da die Variable “$zugang“ den Wert “1“ erhält, also wahr ist. Das wäre eigentlich auch nicht falsch, doch wird hier nicht geprüft, von wo aus die Variable “$zugang“ definiert wurde. Wer sagt und beziehungsweise überprüft, dass dies durch die erste if-Abfrage geschehen ist? Niemand. Somit haben wir schon das erste Sicherheitsleck in unserer Webapplikation. Ein User könnte nämlich nun ganz einfach Folgendes ins Adressfeld seines Webbrowsers eingeben:

```
http://www.domain.com/index.php?zugang=1
```

Sollte dies ein User tun, können wir mithilfe des Scripts ja schnell erkennen was passiert; er gilt als angemeldet und erhält somit Zugang zum geheimen Bereich.

Das Ganze lässt sich natürlich noch steigern. Wir nehmen uns dazu ein neues Beispiel-Script.

```
<?php
$query = “INSERT INTO users VALUES (’$name’, ’$password’, 0)”;
$result = mysql_query($query);
?>
```

Sollten Sie dieses Scriot nicht verstehen, empfiehlt es sich, kurz einen Blick ins Kapitel SQL-Injection zu werfen, welches später behandelt wird, denn wir werden nun genau eine Solche durchführen.

Das oben aufgezeigte Script ist in diesem Fall dafür gedacht, einen neuen Benutzer in der Datenbank zu erfassen. Dazu werden drei Werte gespeichert, Name, Passwort und noch ein Wert der 0 oder 1 sein kann. 0 bedeutet hier, normaler User und 1 steht für Administrator. Somit ist es jetzt unser Ziel einen User zu erstellen, der Administratorenrechte hat. In diesem Fall ist dies kein Problem und wir können unser Ziel durch die folgende Eingabe in die Adressleiste erreichen:

```
http://www.domain.com/index.php?name=Disenchant&password=geheim’,1)#’
```

Der INSERT Befehl, sieht somit wie folgt aus:

```
INSERT INTO users VALUE (’Disenchant’, ’geheim’, 1)#’, 0)
```

Hier wird nach unserem Angriff ein gültiger Query ausgeführt, der einen User mit Administratorenrechten erstellt, da nach # der Rest der Zeile ausgeblendet wird.

Nun wollen wir uns natürlich auch noch in die Rolle des Verteidigers, beziehungsweise des Web-Programmierers oder Betreuers der Webapplikation versetzen und uns überlegen, wie wir uns denn gegen solche leider sehr einfachen auszuführenden Angriffe schützen können.

In der Konfiguration von PHP gibt es zwei Attribute, beziehungsweise Flags, die wir ändern können, um die Sicherheit der Scripts automatisch zu erhöhen. Da wäre zum einen register_globals und zum anderen magic_quotes_gpc.

register_globals

Wenn diese Option aktiviert ist, sind Angriffe, wie der erste in diesem Kapitel, möglich. Es werden alle Variablen automatisch registriert oder anders ausgedrückt, es ist nicht von Interesse, von wo oder wem eine Variable definiert wurde. Somit empfiehlt es sich register_globals zu deaktivieren.

Haben wir dies getan, müsste das erste Script wie folgt aussehen:

```
<?php
      echo $_GET['variable'];
?>
```

So können wir immer angeben, woher eine Variable stammt. In diesem Fall wird sie per GET übergeben und kann auch nicht mehr (so einfach) manipuliert werden. Noch besser wäre es natürlich, wenn wir möglichst alle Variablen per POST übergeben würden.

magic_guotes_gpc

Die zweite wichtige Funktion, wenn es um Sicherheit in PHP-Scripts geht, ist sicher die magic_quotes Funktion. Diese Einstellung wendet die addslashes-Funktion auf alle übertragenen oder ankommenden Daten an. Das heisst genauer gesagt, auf alle GET-, POST-, und auch Session-, sowie Cookievariablen. Diese Funktion setzt vor kritische Stellen/Zeichen, die zur Ausführung von Codes genutzt werden könnten, einfach ein Backslash. Dieser verhindert, dass ein sauberer Befehl hergestellt werden kann, jedoch alle Zeichen so wie sie eingegeben wurden auch wieder ausgegeben werden (wenn dies vom Programmier so gewünscht wird).

Am besten lässt sich dies an der bereits gezeigten SQL-Injection zeigen:

Vorher:
```
INSERT INTO users VALUE ('Disenchant', 'geheim', 1)#', 0)
```

Nachher (mit magic_quotes):
```
INSERT INTO users VALUE ('Disenchant', 'geheim\', 1)#\', 0)
```

Somit wäre der Angriff wirkungslos. Wir sehen, es muss nicht immer kompliziert und schwer sein, Angriffe abzuwehren. Teilweise reicht

es, eine einzige Einstellung zu ändern und fertig ist der ganze Zauber.

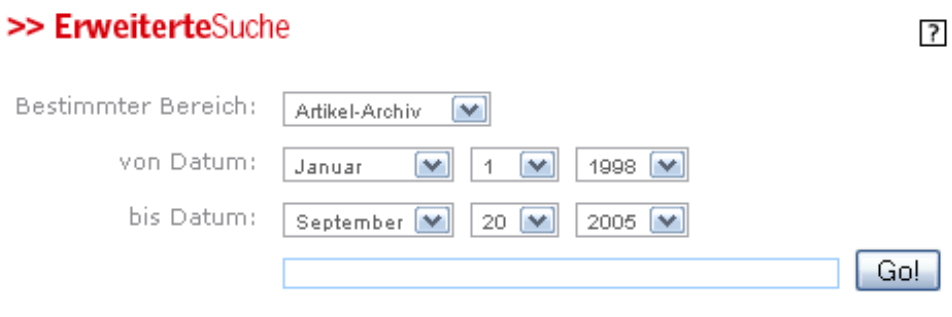
Als kleine Nebeninformation ist hier vielleicht noch zu sagen, dass seit PHP 4.2.0 register_globals standardmässig ausgeschaltet und magic_quotes_gpc eingeschaltet ist. Es gibt es natürlich noch unzählige weitere Mechanismen, die man einbauen könnte, um sich vor Input-validation-Angriffen zu schützen. So könnte man zum Beispiel Variablen noch zusätzlich verschlüsseln oder es wäre möglich alles noch mit Session-Cookies nachzuprüfen, doch wenn ich auf alle diese Massnahmen eingehen würde, könnte ich wohl viele Seiten mehr füllen. Ich bin der Meinung, dass jemand, der eine Webapplikation schreibt, auch immer selbst überlegen muss, wie er seine Applikation schützen will, beziehungsweise, wie viel Schutz diese überhaupt benötigt.

Cross-Site Scripting

Nun beschäftigen wir uns mit spezifischen Problemen von Webapplikationen. Um einen spannenden Einstieg zu bieten, beginne ich gleich mit der zurzeit wohl am meisten im Trend liegenden Angriffsform auf Webapplikationen, dem Cross-Site Scripting oder auch XSS genannt (es wird deswegen nicht CSS genannt, da diese Abkürzung schon für Cascading Style Sheets verwendet wird).

Unter XSS werden alle Angriffe auf Webapplikationen zusammengefasst, mit denen versucht wird, Seiten aus einem anderen Kontext, beziehungsweise von einer anderen Webseite in die angegriffene Seite/Webapplikation zu integrieren. Hierzu ein kleines Beispiel: eine Suchfunktion, deren Eingabe nicht sauber überprüft wird.

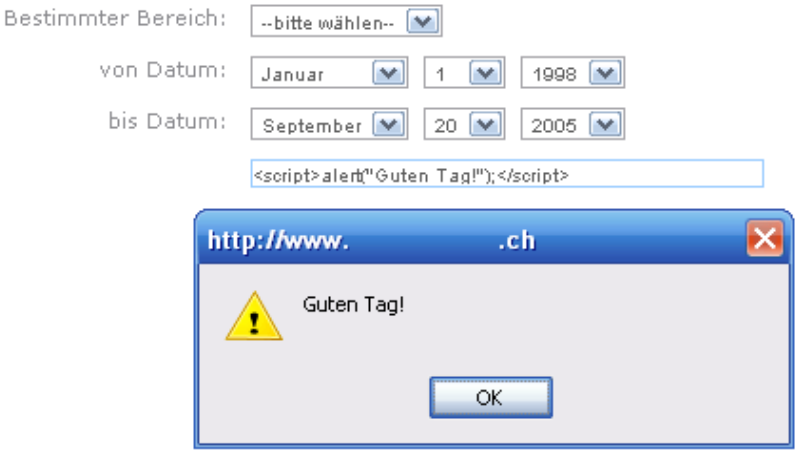
Das Suchformular:



Nun geben wir als Beispiel folgende Suchanfrage an das Formular weiter:

```
<script>alert("Guten Tag!");</script>
```

Nun bestätigen wir die Abfragen mit Go! und das Ergebnis ist Folgendes:



Warum? Ganz einfach: die Suchbegriffe werden nach dem Drücken von Go! wieder an das Formular gesendet, wo die Felder dann gleich gefüllt sind, wie man sie vorher ausgefüllt hat. Soweit so gut, denn das wäre ja eigentlich eine gute Idee. Nur wurde vergessen, dass, wenn dort ausführbarer (Script-)Code eingeschleust wird, dieser dann dadurch, dass er mit der Seite geladen wird, auch zur Ausführung kommt.

Nun stellt sich die Frage, was man denn solchen Angriffen entgegensetzen kann, sowohl in der Position des Web-Programmierers als auch in der des Users. Auf Seiten des Web-Programmierers, steht die sichere und korrekte Programmierung im Vordergrund. Er sollte darauf achten, dass der User möglichst wenig Einfluss auf Variablen hat, die dann in irgendeiner Form weiterverwendet werden. Dazu gehört natürlich auch das Vermeiden von Variablen, die per GET übergeben werden, wo immer dies möglich ist. Des Weiteren sollte er auch darauf bedacht sein, dass, wenn doch einmal eine Variable durch den User veränderbar sein muss, die Eingebe limitiert wird. Um gleich beim Beispiel zu bleiben: es macht keinen Sinn, in einer Suchabfrage Zeichen wie “<“ oder “>“ zuzulassen und es wäre auch sinnvoll, sämtliche Sonderzeichen und auch Umlaute in HTML-Schreibweise umzuwandeln, sowie “ nur noch mit einem Backslash davor zu akzeptieren, also \“.

Die Aufgabe des Users besteht nicht darin, die Webapplikation zu schützen, sondern in erster Linie sich selbst. Dies ist meist gar nicht so einfach, da gute XSS-Attacken eigentlich nicht enttarnt werden können, ohne das jeweilige Fachwissen über die ausgenutzte Fehlprogrammierung in der Webapplikation, welche XSS zulässt. Das heisst, dass der Betreiber einer Webapplikation beziehungsweise der/die Web-Programmierer die volle Verantwortung für den Schutz der Webapplikation vor XSS übernehmen müssen.

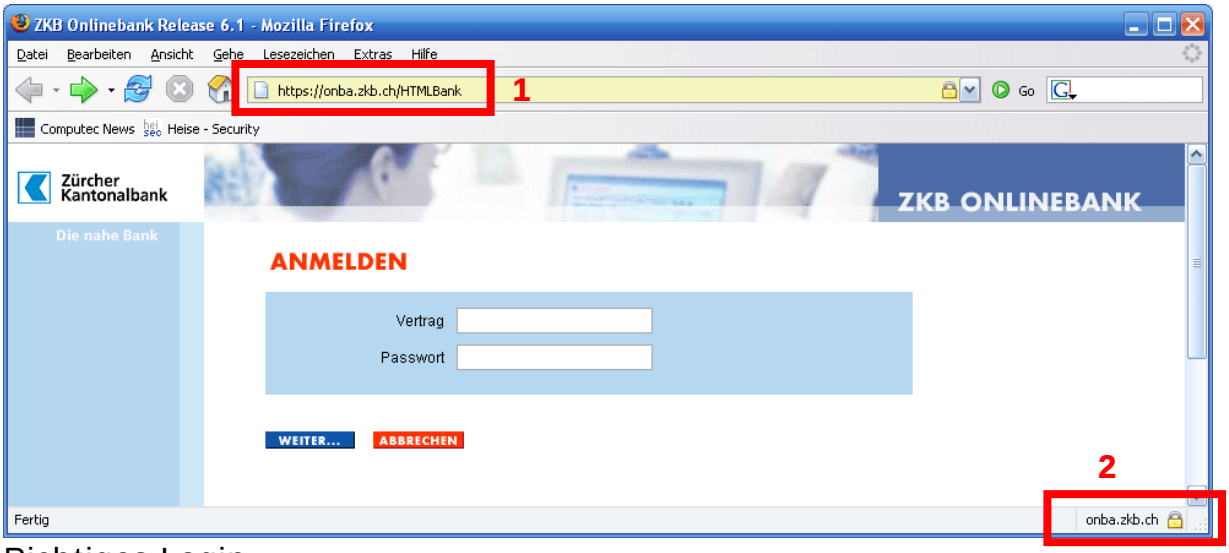
Als Nächstes kommt natürlich, vor allem für die Leute, denen der bisherige Inhalt völlig neu war, die Frage auf: „Was soll den daran gefährlich sein?“ Nun, es gibt unzählige Möglichkeiten, die sich durch XSS-Angriffe umsetzen lassen, was sicher auch, wie schon früher erwähnt, dazu beiträgt, dass diese Art Angriff stark im Trend liegt. Um jedoch die vorherige Frage nicht einfach abzutun, werde ich kurz auf zwei der wichtigsten Gefahren eingehen.

Phishing.

Es lässt sich in die Kategorie “Social Engineering“ einstufen, was soviel heisst wie: „nicht einen Computer zu manipulieren, sondern den User, der davor sitzt“.
Hier finden Sie noch mehr Informationen zum Thema Social Engineering http://www.computec.ch/download.php?view.484.

Es bezeichnet eine Angriffsform, bei der dem User eine (in diesem Fall) nachgebaute Website zeigt, auf welcher der unvorsichtige User zum Beispiel seine Online-Bankig Zugangsdaten eingibt, die dann aber nicht bei der Bank, sondern beim Angreifer, in diesem Fall einem Phisher, landen. Das besondere an Phishing ist, dass der Angreifer zum einen nicht direkt mit dem Opfer in Kontakt tritt, und zum anderen, dass Phishing immer eine Reaktion vom Opfer verlangt, sei es das Besuchen einer Webseite über einen Link aus einem Phishing-Mail oder zum Beispiel die Eingabe von persönlichen Daten auf einer gefälschten Webseite.

Nun möchte ich Ihnen mithilfe eines E-Banking-Logins zeigen, wie Sie als User viele Phishing-Angriffe ohne grossen Aufwand erkennen.



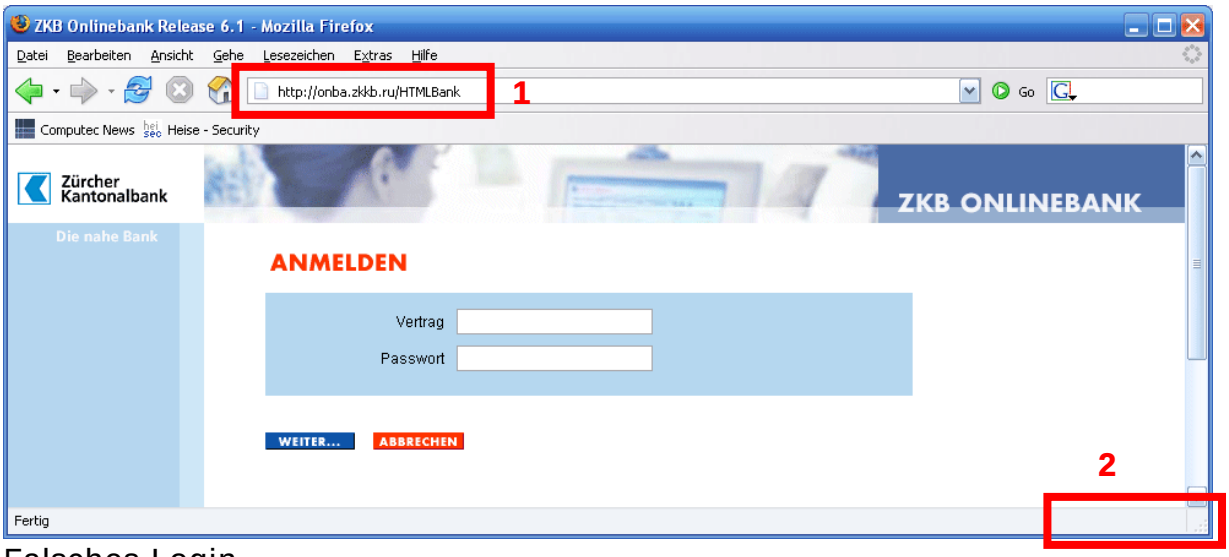
Richtiges Login

Was Sie hier sehen, ist die ursprüngliche Webseite. Es gibt dabei zwei Indizien, die dies bestätigen.

1. Die URL weisst darauf hin, dass es sich um eine gesicherte, beziehungsweise verschlüsselte Verbindung handelt. Dies erkennen wir am Präfix „https://“.

   Weiter können wir an der URL sehen, dass diese auf die richtige Seite verweist. Lassen Sie sich jedoch nicht zu leicht davon täuschen, es gibt viele Tricks, diese URL zu ändern.

2. Das zweite, wohl markanteste und zugleich auch sicherste Indiz dafür, dass Sie sich auf der gewünschten Website befinden, ist das kleine Schloss unten rechts. Es bestätigt, dass die angezeigte Webseite auch die richtige ist, was eine Prüfstelle bestätigt hat. Das Schloss ist somit ein Zeichen für eine Seite, die ein Echtheits-Zertifikat trägt.



Falsches Login

   Was Sie hier sehen ist eine gefälschte Seite, wie sie ein Phisher online stellen würde. Auf den ersten Blick wird man keinen Unterschied zum Original feststellen, doch sehen wir uns nun einmal die beiden vorher erwähnten Punkte an, um die Echtheit der Seite zu überprüfen.

1. In der URL sehen wir, dass auf den ersten Blick eigentlich alles in Ordnung ist, ein zweiter Blick sagt uns jedoch, dass da trotzdem etwas falsch ist.

   Die Seite ist nicht verschlüsselt, was wir am Präfix „http://“ (statt https://) erkennen können.

   Die Angaben in der URL sind falsch, denn zum einen steht anstatt „zkb“ „zkkb“ und zum anderen haben wir bei der Toplevel-Domain

statt „.ch“ für Seiten aus der Schweiz, die Kennung „.ru“ für Russland.

2. Hier sehen wir schnell wo der Fehler liegt, es ist nämlich kein Schloss vorhanden, welches die Echtheit der Webseite bestätigen würde.

   Dies lässt uns darauf schliessen, dass es sich bei dieser Webseite um eine Fälschung handelt und wir auf keinen Fall unsere Daten dort eingeben sollten.

   Wenn Sie sich ein wenig mehr mit dem heutzutage im Internet allgegenwärtigen Thema Phishing befassen wollen, finden Sie dazu ein Dokument aus meiner Feder unter:
   http://www.computec.ch/download.php?view.532

Cookie Hijacking

Das Cookie Hijacking oder auch Cookie Stealing genannt, ist ein weiteres Ziel, das ein Angreifer mit XSS erreichen könnte. Dazu gleich zu Beginn ein kleiner Auszug aus Wikipedia.org, was Cookies überhaupt sind.

„Ein HTTP-Cookie, auch Browser-Cookie oder einfach Cookie genannt, bezeichnet Informationen, die ein Webserver zu einem Browser sendet, die dann der Browser wiederum bei späteren Zugriffen auf denselben Webserver zurücksendet. Mit Cookies ist das zustandslose Hypertext Transfer Protocol um die Möglichkeit erweitert, Information zwischen Aufrufen zu speichern.“

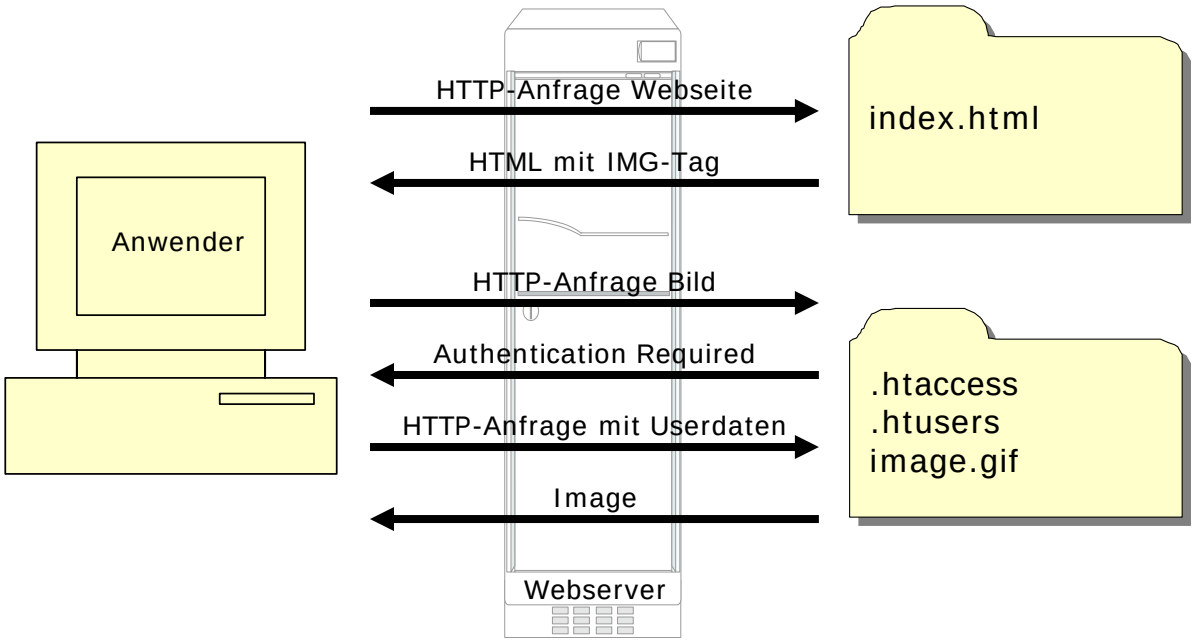
Solche Cookies werden zum Beispiel oft dafür verwendet, um einen User in einem geschützten Bereich einer Webapplikation zu identifizieren, wenn er zwischendurch mal nicht auf der Seite war. Denn wenn Cookies angelegt wurden, kann er einfach wieder in den geschützten Bereich eintreten, aber muss sein Passwort so nicht immer wieder eingeben. Eigentlich eine sehr schlaue Idee, doch was passiert, wenn jemand diese Cookies stehlen kann und sich dann als den besagten User beim geschützten Bereich ausgibt? Um beim Thema XSS zu bleiben und es kurz zu machen: „Mit XSS können solche Cookie Hijackings problemlos durchgeführt werden.“

Cross-Site Authentication (XSA)

Eine neuartige Angriffstechnik, die meines Wissens bis jetzt noch nirgendwo in deutscher Sprache frei verfügbar und zugleich sauber dokumentiert wurde. Somit werde ich dies hier bestmöglich versuchen.

XSA beruht auf dem Einsatz eines geschützten Verzeichnisses auf einem Webserver mit einer so genannten htaccess-Datei. Für Leser, die noch nie mit htaccess zu tun hatten, hier ein kleiner Exkurs, damit auch diese den Ausführungen über XSA folgen können.

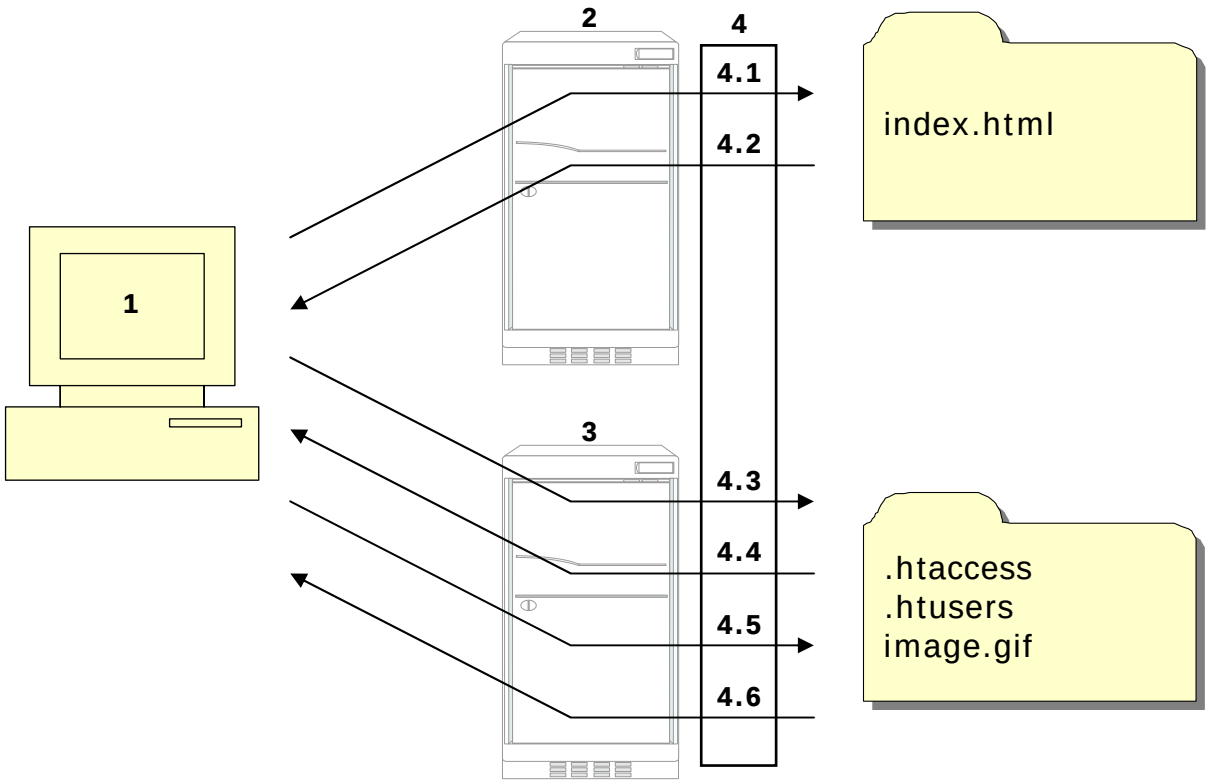
Der Aufbau von einem Verzeichnisschutz mit Hilfe von htaccess beziehungsweise der Zugriff auf das Bild image.gif, sieht wie folgt aus:



Durch die Existenz der Datei .htaccess wird jedem User beim Zugriff auf eine Datei im Ordner der durch die .htaccess geschützt wird verwehrt, jedoch erhält der Besucher eine Anmeldemaske, bei der er sich identifizieren beziehungsweise authentifizieren kann. Wenn der User nun seine Zugangsdaten eingegeben hat, werden diese mit den Daten aus der Datei .htusers verglichen. Wird dort nun ein Eintrag mit den gleichen Daten wie die, welche eingegeben wurden, gefunden, so erhält der Besucher Zugriff auf die gewünschten Ressourcen. Ansonsten bleibt ihm der Zugang weiterhin verwehrt.

Nun jedoch zurück zum Cross-Site Authenticating: Wir haben gesehen wie so ein htaccess-Schutz aufgebaut ist und wie dieser funktioniert. Überlegen wir uns einmal, was denn passieren würde, falls ein Bild auf irgendeinem Webserver in einem mit htaccess geschützten Bereich liegt und wir dieses Bild aber in einer Webseite aufrufen, egal auf welchem Server dieser Welt. Sie ahnen vielleicht schon, was nun passiert. Die Seite

wird geladen und sobald versucht wird, dass Bild zu laden, erscheint die htaccess-Zugangsabfrage.



1.) Anwender
2.) Vertrauenswürdiger Webserver
3.) Webserver des Angreifers
4.) Ablauf der XSA

4.1) HTTP-Anfrage Webseite
4.2) HTML mit IMG-Tag

4.3) HTTP-Anfrage Bild
4.4) Authentication Required
4.5) HTTP-Anfrage mit Userdaten
4.6) Image

Was können wir aber nun mit diesem Wissen erreichen? Wenn wir zum Beispiel irgendwo ein Bild verknüpfen können (in diesem Fall das Bild image.gif aus der Grafik am Anfang von XSA), was heute sehr verbreitet ist, zum Beispiel bei Ebay oder auch in vielen Foren, würde dem User, der einen Foreneintrag mit Verlinkung auf image.gif ansehen möchte, der Zugriff verwehrt. Nun, das ist ja nicht ganz korrekt, denn er erhält einfach eine Maske, an der er sich authentifizieren muss, doch diese

Zugangsdaten hat er ja nicht. Sie werden sich nun sicher fragen, was das denn bringen soll, so sieht er einfach das Bild nicht. Diese Frage werde ich Ihnen nun beantworten. Wir konfigurieren unseren htaccess-Schutz einfach so, dass jede Eingabe als richtig angesehen wird und diese zugleich auf unserem Server in ein Logfile geschrieben wird. Eigentlich würde das schon reichen um einen Grossteil der Benutzer dazu zu bringen, ihre Daten dort einzugeben, denn sie wissen ja nicht, dass der Schutz nur ein Bild und nicht die ganze Seite betrifft. Um aber einen noch grösseren "Erfolg" zu erzielen, empfiehlt sich, noch einen Kommentar in die htaccess-Abfrage einzubauen, der etwa wie folgt lautet: „Ihre Session ist abgelaufen, bitte geben Sie hier Ihre Benutzerdaten erneut ein."

Die einzige Möglichkeit zu erkennen, dass diese Authentifizierungsanfrage von einem anderen Server stammt, ist sich die angezeigte URL anzusehen (hier localhost). Je nach Webbrowser sieht man besser, beziehungsweise schlechter, von welchem Server die Anfrage gestellt wird, doch es ist meistens so angegeben, dass dies vom User nicht bemerkt wird.

Die Erfolgschancen bei einem solchen Angriff sind extrem hoch, da diese Art Angriff noch (fast) nirgendwo eingesetzt wurde und auch erst am 12. Juli 2005 erstmals erwähnt worden ist.

Einen wirklichen Schutz vor XSA gibt es nicht, da es keine eigentliche Fehlfunktion, einer beteiligten Applikation oder eines Verfahrens ist. Es ist somit lediglich möglich, anhanden der falschen URL in der htaccess-Anmeldemaske zu erkennen, ob diese nun echt ist oder nicht.

SQL-Injection

Einem regelrechten Boom erleben Angriffen auf Webapplikationen mit so genannten SQL-Injections. Dabei handelt es sich eigentlich um nichts

Anderes als SQL-Befehle, die (wenn am richtigen Ort eingebunden) verheerende Auswirkungen haben können. Zum Beispiel könnte man mit Hilfe einer SQL-Injection die Abfragen eines Online-Shops so manipulieren, dass bei der Auflistung aller Produkte stattdessen alle User und deren Passwörter (oder Hashes) ausgegeben werden.

Damit nun aber auch die Leser, die SQL (noch) nicht beherrschen, den später folgenden praktischen Beispielen folgen können, werde ich hier kurz die Grundlagen dieser Datenbanksprache erklären. Da wir aber nicht SQL als Hauptthema behandeln und es darüber genügend gute und frei verfügbare Literatur gibt, kann ich nicht auch noch auf die Struktur von Datenbanken eingehen.

Als Ausgangslage nehmen wir eine Tabelle die wie folgt aussieht (normalerweise gehört eine UID auch dazu, aber ich finde es hier überflüssig):

name	**passwort**	**wohnort**	**admin**
Hans	Admin123	Zürich	1
Fredi	sagichnicht	Bern	0

SELECT

Eines der wichtigsten Dinge bei einer Datenbank ist das Auslesen von zuvor darin gespeicherten Daten, in SQL heisst dieses Auslesen SELECT. Eine Abfrage an die Datenbank könnte etwa wie folgt aussehen:

```
SELECT name, passwort FROM user;
```

Hierbei werden alle Daten aus den Spalten name sowie passwort ausgelesen. Diese beiden Spalten befinden sich in der Tabelle namens user. In unserer Demo-Tabelle würde somit Folgendes abgefragt werden:

name	**passwort**	**wohnort**	**admin**
Hans	Admin123	Zürich	1
Fredi	sagichnicht	Bern	0

Wollen wie die Abfrage spezifizieren, können wir zusätzlich noch die WHERE-Anweisung verwenden.

```
SELECT name, passwort FROM user WHERE wohnort='Zürich';
```

Wenn wir die Abfrage so erweitern, erhalten wir nicht mehr wie vorher alle Datensätze aus der betroffenen Tabelle sondern nur die, welche in ihrer Spalte beim Attribut wohnort Zürich eingetragen haben.

name	passwort	wohnort	admin
Hans	Admin123	Zürich	1
Fredi	sagichnicht	Bern	0

Meist wird nun an dieser Stelle auf den Befehl UNION SELECT eingegangen, der mehrere Abfragen miteinander verknüpfen kann, doch halte ich es für besser, diesen erst dann einzuführen, wenn wir in auch praktisch einsetzen können und dies ist in diesem Dokument nicht der Fall. Sollte es jedoch eine Version 2.0 geben, wird dies sicher vorhanden sein.

INSERT, UPDATE und DELETE

Kommen wir nun zur Manipulation von Datensätzen mit der SQL. Wer ein wenig Englisch beherrscht, dem muss eigentlich nicht erklärt werden, was den diese Befehle für Auswirkungen haben. Ich werde dies aber dennoch tun und auch zeigen, wie diese in SQL eingesetzt werden können. Beginnen wir mit dem Einfügen eines neuen Datensatzes.

```
INSERT INTO user (name, passwort, wohnort, admin) VALUES ('Disenchant',
'geheim', 'Frauenfeld', 1);
```

Mit diesem Befehl, bringen wir die Datenbank dazu, in die Tabelle user einen neuen Benutzer zu erstellen, der den Namen Disenchant hat, das Passwort geheim und mit der Angabe, dass er in Frauenfeld wohnhaft ist.

name	passwort	wohnort	admin
Hans	Admin123	Zürich	1
Fredi	sagichnicht	Bern	0
Disenchant	geheim	Frauenfeld	1

Wir sehen, SQL hat einen sehr einfachen und leicht verständlichen Syntax, der schnell verständlich ist. Um wieder mal ein Wort über das eigentliche Thema dieses Dokuments zu verlieren, wollen wir uns überlegen, was wir nun zum Beispiel mit dem Befehl INSERT tun könnten, wenn wir die Datenbank einer Webapplikation dazu bringen könnten, diesen auszuführen. Als einfaches Beispiel könnten wir, wie bereits gemacht, einen User erstellen, der in einem Feld namens admin den Wert 1 beinhaltet (das würde in diesem Fall bedeuten, dass der

neu erstellte User ein Administrator ist). So könnten wir ohne weiteres einen neunen Administrator erstellen.

Machen wir weiter mit einer weiteren Form von Manipulation an der Datenbank. Diesmal mit dem Befehl UPDATE, welcher im Gegensatz zu INSERT keinen neuen Datensatz erstellt, sondern einen bestehenden verändert, beziehungsweise einzelne Teile davon überschreibt.

```
UPDATE user SET wohnort='Luzern', admin='0' WHERE name='Disenchant';
```

Um uns das Ganze vorzustellen, denken wir zum Beispiel daran, ich wäre umgezogen und hätte darum nur noch Leseberechtigung auf der Datenbank der Webapplikation. Die Tabelle user würde dann wie folgt aussehen.

name	**passwort**	**wohnort**	**admin**
Hans	Admin123	Zürich	1
Fredi	sagichnicht	Bern	0
Disenchant	geheim	Luzern	0

Diesen Befehl UPDATE könnten wir zum Beispiel auch verwenden, um aus einem normalen User einen Administrator zu machen. Kommen wir nun aber noch zu DELETE.

```
DELETE FROM user WHERE name='Fredi';
```

Damit würden wir den Datensatz beziehungsweise alle Datensätze löschen, bei welchem/welchen in der Spalte name der Wert Fredi steht. Auf unsere DEMO-Tabelle hätte dies folgende Auswirkung:

name	**passwort**	**wohnort**	**admin**
Hans	Admin123	Zürich	1
Disenchant	geheim	Luzern	0

Wir sehen, der User Fredi wurde gelöscht.

Wollen wir uns nach diesem kleinen Exkurs nun aber den Angriffen widmen, die man mit diesem Wissen über SQL umsetzen kann; den so genannten SQL-Injections.

EIGENTLICHES THEMA SQL-INJECTION

SQL-Injections sind zurzeit wohl mit XSS zusammen die am häufigsten gefundenen und ausgenutzten Schwachstellen in Webapplikationen. SQL-Injections sind unter anderem darum so gefährlich, da fast alle modernen

webbasierenden Projekte auf eine Datenbank zurückgreifen und genau diese wird hier angegriffen.

Die wichtigsten Werkzeuge für eine SQL-Injection

Steuerzeichen	Bedeutung für die SQL-Injection
'	Durch das Einfügen eines einfachen Anführungszeichens kann meist schon überprüft werden, ob eine Webapplikation durch SQL-Injections potenziell gefährdet ist.
#	Der Rest der Zeile wird auskommentiert.
/*	Alles was danach noch folgt, wird auskommentiert.
OR 1=1 OR 1='1 OR 1="1	Damit lässt sich ein Statement auf true setzen.

Mit diesem Grundwerkzeug und dem vorhergegangenen kleinen Exkurs zum Thema SQL, ist es Ihnen nun möglich SQL-Injections in der Praxis umzusetzen.

SQL-Injection in der Praxis

Wir sind beim Thema Input-validation-Angriffe schon auf ein kleines Beispiel einer SQL-Injection eingegangen. Daher möchte ich hier ein wenig weiter gehen und ein zwar noch nicht als schwierig einzustufenden Angriff zu zeigen, jedoch einen sehr praxisnahen, da wir ein Login-Script verwenden werden, wie dies bei vielen Webapplikationen üblich ist. Als Information ist zu sagen, dass dafür zwingend magic_quotes_gpc in der php.ini auf OFF gestellt werden muss.

Als Erstes brachen wir natürlich das besagte Script:

```
<html>
<head><title>SQL-Injection - Insecure Login-Script</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head>
<body>

<?php
      if (!empty($username)) {
            $link = mysql_connect("localhost", "root", "");
            if (!$link)
                  die("DB Verbindungsfehler");
            if (!mysql_select_db("sql_injection", $link))
                  die("DB Auswahlfehler");

            $query = "SELECT * FROM `userlist` WHERE `username` =
            '$username' AND `password` = '$password'";

            $result = mysql_query($query, $link);
```

```
            if (!$result)     {
                  print mysql_error();
                        die("<p>Es ist ein Fehler aufgetreten!</p>");
            }

            while ($array = mysql_fetch_array($result)) {
              $logged_in = 'ja';
              $username = $array[username];
              $password = $array[password];
            }

            if ($logged_in == 'ja') {
                  echo "Hallo $username, dein PW ist $password<br />";
                } else {
                echo "Fehlerhafte Eingaben<br />";
            }
            mysql_close($link);


      } else {
      echo "Login<br>
            <form name=\"login\" method=\"post\" action=\"\">
            <p>Benutzername<br /><input type=\"text\" name=\"username\"
            size=30><br />
            <p>Passwort<br /><input type=\"password\" name=\"password\"
            size=30></p>
            <input type=\"submit\" value=\"Login\">
            </form>";
}
?>

</body>
</html>
```

Dann erstellen wir eine MySQL-Datenbank mit dem Namen sql_injection und führen darin (zum Beispiel mit phpMyAdmin) folgenden SQL-Befehl aus.

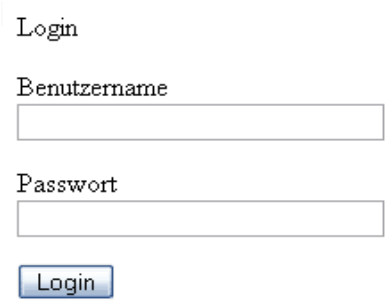
```
CREATE TABLE `userlist` (
  `username` text NOT NULL,
  `password` text NOT NULL
) TYPE=MyISAM;

INSERT INTO `userlist` VALUES ('admin', 'geheim');
```

Wenn wir auch dies erledigt haben, sollten wir eine in der Datenbank sql_injection eine Tabelle namens userlist haben, die wie folgt aussieht.

username	**password**
admin	geheim

Nun sind wir soweit, dass wir eine praxisnahe SQL-Injection simulieren können. Rufen wir einmal unser Login-Script im Webbrowser auf.



Wir sehen ein normales Formular, in welchem wir unsere Benutzerdaten eintragen könnten, um uns einzuloggen. Da wir ja lernen möchten und unser Wissen auch bei Scripts, die wir später einmal antreffen, einsetzen wollen, greifen wir nun zuerst einmal auf unsere vorhergegangene Tabelle mit den wichtigsten Werkzeugen für eine SQL-Injection zurück. Wir finden darin unter anderem das einfache Anführungszeichen (') mit dem wir in einer Vielzahl von Fällen die potenzielle Verwundbarkeit einer Webapplikation betreffend SQL-Injections testen können. Dies machen wir nun, indem wir einfach bei Benutzername ein einfaches Anführungszeichen einfügen und dann aud Login klicken.

Wir erhalte eine Fehlermeldung, die wie folgt aussehen sollte:

```
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 '''' AND
`password` = ''' at line 1
```

Dies zeigt uns nun mit grosser Sicherheit, dass eine SQL-Injection möglich ist. Somit beginnen wir nun mit dem eigentlichen Angriff.

Sehen wir uns einmal den Query für die Überprüfung der Benutzerdaten genauer an:

```
$query = "SELECT * FROM `userlist` WHERE `username` = '$username' AND
`password` = '$password'";
```

Wir sehen, die Variablen $username und $password werden ohne eine weitere Prüfung direkt in die Abfrage eingefügt. Dadurch erkennen wir auch, warum der Query bei unserem Test auf eine SQL-Injection-Verwundbarkeit eine Fehlermeldung ausgab, denn da sah er so aus:

```
$query = "SELECT * FROM `userlist` WHERE `username` = ''' AND
`password` = ''";
```

Mit diesem Wissen können wir nun natürlich noch mehr erreichen. Sehen wir uns dazu nochmals unsere Werkzeuge aus der Tabelle an, die wir uns für SQL-Injections angefertigt haben. Dort werden wir „/*“ finden, womit wir alles Nachstehende ausblenden können, was wir schon bald ausnützen werden.

Wir wissen ja, da wir die Tabelle selbst erstellt haben, dass es einen Benutzer namens admin gibt (davon kann aber meist ausgegangen werden), somit werden wir diesen Account einmal angreifen. Als Benutzernamen geben wir selbstverständlich admin ein, doch kennen wir als Angreifer das Passwort ja nicht. Hier kommt nun „/*“ zum Zuge. Geben wir admin'/* in das Feld für den Benutzernamen ein und klicken auf Login. Siehe da, wir sind eingeloggt und erhalten die Meldung: „Hallo admin, dein PW ist secret“. Wie hat das nun funktioniert? Sehen wir uns dazu einmal die Abfrage an, die wir durch diese Eingabe produziert haben.

```
$query = "SELECT * FROM `userlist` WHERE `username` = 'admin'/*' AND
`password` = ''";
```

Das heisst auf gut Deutsch: „Selektiere alles aus der Tabelle userlist, in welchem das Attribut username gleich admin ist.“ und damit fertig. Somit wird die ganze Abfrage nach dem Passwort durch „/*“ ausgeblendet. Damit haben Sie ihre erste erfolgreiche SQL-Injection hinter sich.

Das Verhindern solcher teilweise verheerenden SQL-Injections ist für einfachere Angriffe gar nicht sonderlich schwer. Wie schon im Kapitel Input-Validation Angriffe erwähnt gibt es dafür eine Funktion von PHP die auch standardmässig auf ON geschaltet ist, nämlich magic_quotes_gpc. Da diese aber schon besprochen wurde, werde ich nicht nochmals darauf eingehen.

Schlusswort

Damit kommen wir langsam, aber sicher zum Ende dieses Dokuments und natürlich wurde nicht einmal ansatzweise alles darin behandelt, was es an sicherheitsrelevanten Informationen zum Thema Webapplikationen gibt. Sie sollten nun aber ein Verständnis und Gespür dafür haben, worauf Sie als Web-Programmierer, wie auch als User achten müssen um Ihr Risiko im Cyberspace auf ein Minimum zu reduzieren.

Sollte ich ein positives Feedback auf dieses Dokument erhalten, werde ich wenn möglich eine erweiterte Version 2.0 schreiben, die noch mehr Angriffs- und vor allem auch Verteidigungsszenarien aufzeigt. Vor allem ein Bereicht Intrusion-Detection-Systems (IDS) im Bereich Webapplikationen wäre sicher sehr interessant, da es dazu meines Wissens noch sehr wenig Informationen gibt, welche vor allem für Web-Programmierer interessant und hilfreich sind.

Ich spiele zurzeit auch mit der Überlegung, das Dokument in ein offenes E-Book Projekt zu verwandeln, damit jeder seinen Teil dazu beitragen kann und es auch immer aktuell bleibt aber so etwas steht noch in den Sternen.

Danksagungen

Als Erstes möchte ich natürlich Ihnen, liebe/r LeserIn, danken, dass Sie mein Dokument gelesen haben. Dies bedeutet mir natürlich sehr viel, denn es zeigt mir, dass ich mir nicht umsonst die Mühe gemacht habe, dieses Dokument zu schreiben und es gibt mir auch das Gefühl, dass ich helfen konnte.

Ein besonderer Dank gebührt an dieser Stelle auch dem Magazin hakin9, welches für einen Teil dieses Dokumentes als Vorlage gedient, beziehungsweise, wichtige Informationen geliefert hat, die hier wieder eingeflossen sind. Unter anderem sind einige Beispiel-Scripts aus denen von hackin9 entstanden.
http://www.haking.pl/

Ein weiteres Dankeschön geht an die Autoren des OWASPGuide, dem Meisterwerk zum Thema Webapplication-Security, aus dem unter anderem auch ich eine Menge gelernt habe was diesen Bereich betrifft.
http://www.owasp.org/

Zuletzt möchte ich mich natürlich auch noch bei Gérard Mathiuet für die Korrektur bedanken.

Zum Autor

Sven Vetsch ist 1986 geboren und beschäftigt sich seit seinem zwölften Lebensjahr mit der Thematik IT-Sicherheit. Zurzeit arbeitet er in seinem Praktikumslehrjahr zum Mediamatiker mit eidgenössischer technischer Berufsmatura als Technical Assistant für die Schule für Beruf und Weiterbildung (SBW) in Romanshorn. In seiner Ausbildung hatte er schon sehr viel mit Webapplikationen zu tun und hat auch selbst solche programmiert, was schlussendlich auch ausschlaggebend war, dieses Dokument zum Thema Webapplication-Security zu verfassen. Wenn alles wie geplant verläuft, wird Sven Vetsch nach dem Sommer 2006 ein Studium an einer Fachhochschule im Informatikbereich angehen oder sich die eidgenössische Matura erarbeiten, um danach eine Universität zu besuchen. Von Sven Vetsch sind bereits mehrere Publikationen im Computersicherheitsbereich bekannt, die vor allem im Bereich “Social Hacking“ anzusiedeln sind. Für mehr Informationen besuchen Sie die Webseite von Sven Vetsch (www.disenchant.ch).

Mit freundlichen Grüssen
Sven Vetsch