

# Sicherheit von Web-Applikationen aus Sicht eines Angreifers

Tatwerkzeug: Web-Browser

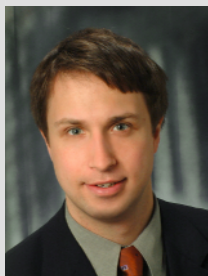
Micha Borrmann, Sebastian Schreiber

*Trotz durchdachtem Patchmanagement, gut gewarteten Sicherheitsvorkehrungen (Firewalls, etc.): Alle Sicherheit ist trügerisch. Schuld daran sind weit verbreitete Fehler in Web-Applikationen.*



Micha Borrmann

Security Consultant beschäftigt bei der SySS GmbH; prüft seit mehr als vier Jahren Web-Applikationen im Kundenauftrag



Sebastian Schreiber

Gründer und Geschäftsführer der SySS GmbH

E-Mail:  
[borrmann@syss.de],  
[schreiber@syss.de]

## Einleitung

Im vergangenen Jahrzehnt haben zahlreiche Unternehmen ihre Systeme und Netzwerke aus dem Internet sinnvoll durch Firewalls geschützt, ein funktionstüchtiges Patchmanagement implementiert und auch Intrusion-Detection-Systeme (IDS) eingesetzt. Die verwendeten Produkte und Prozesse sind weitgehend ausgereift. Aufgrund dieser Investitionen gehen die Verantwortlichen davon aus, die IT-Risiken zu kennen und ein sinnvolles Sicherheitsniveau erreicht zu haben. Dies kann jedoch ein fataler Irrtum sein. Es gibt Schwachstellen in Software, die von den üblichen Schutzmaßnahmen weder erkannt noch verhindert werden und durch die trotz der Sicherheitsvorkehrungen erhebliche Schäden entstehen können.

Dieser Artikel beschreibt derartige Schwachstellen bei webbasierten Anwendungen, da diese für unterschiedlichste Aufgaben eingesetzt werden und trotz aktuell gepatchten Systemen, Firewall- und IDS-Einsatz gravierende Sicherheitsprobleme für ein Unternehmen bedeuten können.

## 1 Häufige Fehler bei Web-Applikationen

Ob (E-Mail-)Kommunikation, Bestellungen von Waren und Dienstleistungen aller Art, Personalrekrutierung oder B2B-Service, Web-Applikationen sind in praktisch allen Bereichen weit verbreitet oder im starken Wachstum. In einer aktuellen Gartner Studie (siehe [Gartner2006]) wird die These vertreten, dass in drei Jahren das (Web-)Portal eines Unternehmens die primäre Benutzerumgebung darstellen wird und so an die Stelle des momentanen Desktop-Systems (inkl. Microsoft Office) tritt. Da die Web-

Applikationen ausnahmslos an das jeweilige Unternehmen angepasst oder komplette Eigenentwicklungen sind, können die Sicherheitsschwächen meist nur bei einer spezifischen Anwendung zu finden sein. Alle folgenden Beschreibungen sind anonymisiert aber von den Autoren bei diversen Web-Applikationen gefunden worden.

### 1.1 URL-Manipulationen

Eine triviale, aber (aus Sicht eines Angreifers) effektive Schwachstelle sind URL-Manipulationen. Teile der URL (Parameter) werden vom Nutzer einer Web-Applikation manipuliert. Exemplarisch kann dies anhand der fiktiven URL <http://www.bank.de/homebanking?konto=0123456> gezeigt werden. Sollte eine Online-Banking-Anwendung entsprechende URLs verwenden, so können die Benutzer einfach den Wert für „konto“ manipulieren, **nachdem** sie sich erfolgreich authentifiziert haben. Wenn dann durch die Manipulation des Parameters „konto“ dem Benutzer Daten Dritter von der Web-Applikation zur Verfügung gestellt werden, so besteht ein erhebliches Sicherheitsproblem. Dies ist beispielsweise für eine Online-Vertragsverwaltung der Deutschen Telekom AG dokumentiert worden, wo durch die Variation des URL Parameters – und zwar der Vertragsnummer – die Verträge aller Benutzer eingesehen und modifiziert werden konnten (siehe [CCC2004]).

Ein ähnlich geartetes Problem besteht durch dynamisch generierte Framesets bzw. Webseiten. Exemplarisch wird die Anmeldemaske der Online-Banking-Software unter <http://www.bank.de/frameset.cgi?userlogin.html> angeboten. Das Frameset der fiktiven Anwendung wird über den Dateinamen angegeben. Benutzer können (und werden)

versuchen, URLs wie beispielsweise <http://www.bank.de/frameset.cgi?etc/passwd> aufzurufen. Sollte dann eine Benutzerliste ausgegeben werden, so liegt im Gesamtsystem wiederum ein schweres Sicherheitsproblem vor. Beliebige Dateizugriffe auf das zugrundeliegende UNIX-Hostsystem sind möglich (UNIX-Systeme speichern die Liste der Benutzer in der Datei /etc/passwd). Ist der Parameter relativ anzugeben (d.h. „.../.../.../etc/passwd“), so wird eine derartige Schwachstelle „Path-Traversal“ genannt.

Weiterhin kann der Parameter durch eine URL manipuliert werden, im Beispiel zu <http://www.bank.de/frameset.cgi?http://www.hacker.com/save.php>. Falls die Anwendung eine entsprechende Schwachstelle aufweist, so wird in dem Frameset des Nutzers ein Teil von einer dritten Seite dargestellt, da ein Web-Browser Elemente einer Seite von unterschiedlichen Servern beziehen kann. Derartige Schwachstellen werden bereits für „intelligenteres Phishing“ verwendet (siehe [Paypal2006]). Die fremde Seite wird optisch dem Original nachempfunden und der Parameter zur Verschleierung kodiert (siehe dazu Abschnitt 3). Die resultierende URL <http://www.bank.de/frameset.cgi?%68%74%74%60%3A%2F%2F%77%77%77%2E%68%61%63%65%72%2E%63%6F%6D%2F%73%61%76%65%2E%70%68%70> wird dann per E-Mail verteilt. Die Benutzer werden dieser URL eine höhere Vertrauenswürdigkeit entgegenbringen, da die URL mit dem System der Bank direkt in Verbindung gebracht wird.

## 1.2 Command-Injection

Über die Manipulation von Parametern lassen sich mitunter auch Systemkommandos einschleusen, die dann vom Webserver ausgeführt werden. Die Ausgabe der Kommandos wird im Web-Browser dargestellt. Dies ist beispielsweise für ältere Versionen der Forums-Software phpBB [1] dokumentiert. Wie in Abbildung 1 zu sehen, wurde der Parameter „highlight“ manipuliert. Die Eingabe „see%2527%252esystem(1s)%252e%2527“ führt auf dem System das Kommando „ls“ aus, welches zum Anzeigen der Dateien verwendet wird. Falls die Web-Applikation auf dem Web-Server administrative Rechte hat, könnten auf diese Art auch neue Benutzer hinzugefügt oder Fest-



Abb. 1: erfolgreiche Command-Injection bei phpBB

platten formatiert werden.

## 1.3 SQL-Injection

Bereits seit vielen Jahren ist diese Problematik bekannt (siehe z.B. [SQL2001] und [SQL2002]). Über Eingabefelder der Benutzer (z.B. Konto-Nummern bei Überweisungsfeldern) lassen sich Datenbankanweisungen in SQL integrieren, die dann vom Datenbank-System ausgewertet werden. Derartige Eingaben sind mit einem Meta-Charakter (meist das Hochkomma „‘“) zu versehen. Um eine korrekte Datenbankanfrage durchführen zu können, muss ein Angreifer Informationen über das Datenbanksystem als auch über den Aufbau der Datenbank selbst haben. Da häufig fehlerhafte Eingaben (beispielsweise ein Hochkomma als einzige Benutzereingabe) detaillierte Fehlermeldungen des Systems weitergeben, erhält ein Angreifer diese Informationen oft sehr einfach (Abbildung 2).

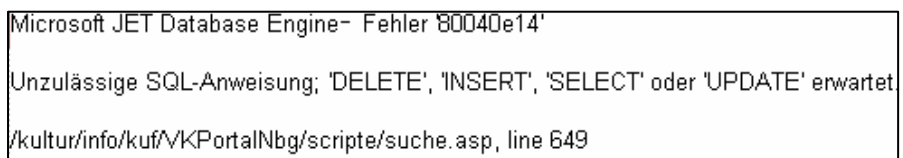


Abb. 2: typische Fehlermeldung die auf eine SQL-Injection-Schwäche hinweist

Da zahlreiche Web-Applikationen eine Authentifizierung mittels Benutzername und Kennwort erfordern und diese Daten in SQL-Datenbanken hinterlegt sind, ist der

Authentifizierungsprozess prinzipiell angreifbar. Mit folgendem „Benutzernamen“ und einem beliebigen Kennwort gelang den Autoren wiederholt eine erfolgreiche Authentifizierung: „meier' OR '1'='1“. Möglicherweise ist bei diesen verwundbaren Web-Applikationen serverseitig folgender SQL-Code vorhanden (ein Angreifer kennt den Code der Web-Applikation gewöhnlich nicht):

```
SELECT permission FROM userdb
WHERE user=' $benutzername'
AND password=' $kennwort'
```

Durch die fehlende Prüfung der Eingabedaten wird serverseitig dadurch das folgende SQL-Kommando ausgeführt:

```
SELECT permission FROM userdb
WHERE user='meier' OR '1'='1'
AND password=' $kennwort'
```

Aufgrund der mathematischen Logik ist dieser Ausdruck stets wahr, d.h. das Kennwort wird zur Authentifizierung nicht benötigt und die Kenntnis eines korrekten Benutzernamens genügt.

SQL-Injections sind prinzipiell auch als URL-Manipulationen durchführbar, wenn die Parameter der URL von einem Datenbanksystem verarbeitet werden. Neben dem

Auslesen (und Manipulieren) Daten Dritter sind auch Denial-of-Service-Angriffe mögliche wenn Daten gelöscht werden.

## 1.4 Cross-Site-Scripting

Obwohl dieses Sicherheitsproblem ebenfalls seit vielen Jahren bekannt ist, wurden derartige Schwächen vor wenigen Jahren kaum ernst genommen. Ein Bankkunde kann beispielsweise seine persönlichen Daten innerhalb der Web-Applikation modifizieren. Bei „Straße“ trägt er „<H1>Goethestr. 3</H1>“ ein. Dieses Einschleusen von HTML-Code wird HTML-Injection genannt. Wenn der Benutzer seine Daten ansieht, so sind diese gemäß seinen Eingaben optisch dargestellt (die Kennzeichnung mit „<H1>“ bewirkt eine Formatierung als Überschrift). Es ist jedoch u.U. möglich, auch Javascript-Code einzuschleusen. Diese Benutzerdaten werden von der Web-Applikation gespeichert. Wer sich die Daten ansieht (gewöhnlich nur der legitimierte Benutzer), in dessen Web-Browser wird dann der entsprechende Code ausgeführt, und zwar im Kontext der Webseite der Bank! Die persönlichen Daten werden jedoch gelegentlich auch von Kundenberatern des Kreditinstitutes aufgerufen, und diese werden der Webseite ihres Arbeitgebers Vertrauen entgegenbringen. Dadurch lassen sich über eine Webseite Daten Dritter einsehen, wenn eine Anwendung das Speichern von Daten gestattet und HTML- bzw. Javascript-Injection-Schwächen bestehen.

Doch auch mittels URL-Manipulationen sind Cross-Site-Scripting-Attacken möglich, wenn Parameter in der Darstellung der Seite Verwendung finden. Ein Angreifer muss dann eine entsprechende URL verteilen (z.B. per E-Mail). Diese Angriffsart ähnelt sehr dem in Abschnitt 1.1 beschriebenen „intelligenterem Phishing“.

## 1.5 Sessions

Da HTTP ein zustandsloses Protokoll (*stateless protocol*) ist, müssen Maßnahmen integriert werden, um zusammengehörige Anfragen zu erkennen. Denn das HyperText Transfer Protokoll (HTTP) kennt – im Gegensatz beispielsweise zum Post Office Protokoll (POP3) – keine Benutzeran- und abmeldung. Dazu werden bei Web-Applikationen Sessions in HTTP integriert. Dies erfolgt durch ein Token, das vom Server beim Start der Anwendung, d.h. z.B. nach erfolgreichem Senden gültiger Authentifizierungsdaten, an den Client (Web-Browser) übermittelt und danach bei jedem weiteren Kommunikationsvorgang bidirek-

tional mitgesendet wird. Das Token wird Session-ID genannt, denn es repräsentiert die Session bzw. wird mit einer spezifischen Session identifiziert. Die Session-ID („sid“) wird beispielsweise in der URL übertragen, z.B.

<http://www.bank.de/index.jsp?sid=123456>. Werden inkrementelle (oder andere vorher-sagbare) Werte für die Session-ID verwendet, so besteht ein hohes Risiko, dass Dritte Zugang zu einer bestehenden Session erhalten. Ferner weisen Web-Applikationen oft Schwächen beim Abmelden von Benutzern auf, wenn beispielsweise die Funktionalität „Abmelden“ lediglich einen Link zu <http://www.bank.de> darstellt. Ein erneutes Anmelden ist damit möglich und die nicht-authentifizierte Nutzung der Webseite ebenfalls. Doch serverseitig kann die mit der Session-ID verknüpfte Session noch verwendet werden, u.U. noch mehrere Wochen lang.

## 1.6 Zusammenfassung

Die beschriebenen Fehler in Web-Applikationen sind nicht durch klassische Firewall-Technologie erkenn- oder gar vermeidbar. Die möglichen Schäden umfassen:

- Datenausspähung
- Unerlaubte Datenmanipulation
- Identitätsdiebstahl
- Denial-of-Service

Praktisch alle diese Gefahren nutzen lediglich zwei Schwachstellen aus:

- Unvalidated Input (siehe [TopTen])
- Session Management

Da URL-Manipulationen, Command- & SQL-Injections sowie Cross-Site-Scripting auf Benutzereingaben basieren, die von den (Web-) Anwendungen nicht (bzw. nicht ausreichend) validiert wurden, lassen sich diese Fehler zu einer Schwachstelle zusammenfassen. Die eigentlichen Ursachen dafür, dass derartige Schwachstellen in produktiven Anwendungen enthalten sind, ist einerseits das häufig unzureichende KnowHow der Software-Entwickler, die nicht selten erstaunt sind, was für Schwachstellen die von ihnen entwickelten Anwendungen aufweisen. Andererseits werden diese Schwachstellen aufgrund einer mangelhaften Qualitätskontrolle im Softwareentwicklungsprozess nicht identifiziert.

Defizite im KnowHow der Software-Entwickler als auch im Qualitätsmanagement des Entwicklungsprozesses sind die eigentlichen Ursachen der Sicherheitsprobleme.

## 2 Finden von Fehlern

Um Fehler in Web-Applikationen zu ermitteln, werden die zwei generellen Schwachstellen überprüft.

Um das Session Management zu testen, sind üblicherweise zahlreiche An- und Abmeldevorgänge notwendig. Zur Durchführung mehrerer tausend Anmeldevorgänge werden meist individuelle Scripte (z.B. in Perl geschrieben) verwendet. Dazu wird für einen Anmeldevorgang der Netzwerkverkehr protokolliert, z.B. mit Wireshark [2]. Anschließend wird der eigentliche Anmeldevorgang – häufig ein HTTP-POST-Request – isoliert und automatisiert wiederholt. Von Interesse ist die Vergabe der Session-ID in der Antwort der Web-Applikation auf den Anmeldevorgang. Es soll festgestellt werden, ob die Session-IDs zufällig oder vorhersagbar sind. Dazu ist die Session-ID in einen numerischen Wert zu überführen, falls nicht nur Ziffern für die Session-ID verwendet werden. Sollten lediglich die Zeichen 0-9 und A-F verwendet werden, so handelt es sich um Hexadezimalzahlen, die in Dezimalzahlen konvertiert werden können. Für alle sonstigen Zeichensätze von Session-IDs kann das Script `charset.pl` [3] zur Umwandlung in Dezimalzahlen verwendet werden. Die auf diese Art und Weise erhaltenen numerischen Session-IDs werden auf Auffälligkeiten geprüft. Kontinuierliche Sprünge in den Zahlenreihen bedeuten eine prinzipielle Vorhersagbarkeit. Das Fehlen dieser ist in einer grafischen Darstellung am „weißen Rauschen“ erkennbar. Dazu werden die erhaltenen Session-IDs als Messwerte aufgefasst und chronologisch dargestellt (z.B. mit Gnuplot [4]; siehe Abbildung 3).

Falls die Daten auf dem Übertragungsweg mittels TLS geschützt sind („HTTPS“), kann die Protokollierung nicht mit einem Sniffer erfolgen, da die Daten verschlüsselt übertragen werden. Entweder wird die Protokollierung der Daten im Browser durchgeführt (z.B. durch die Mozilla Extension LiveHTTPHeaders [5]), oder es wird ein HTTPS zu HTTP Konverter eingesetzt (z.B. Stunnel [6]) und der unverschlüsselte Datenstrom analysiert.

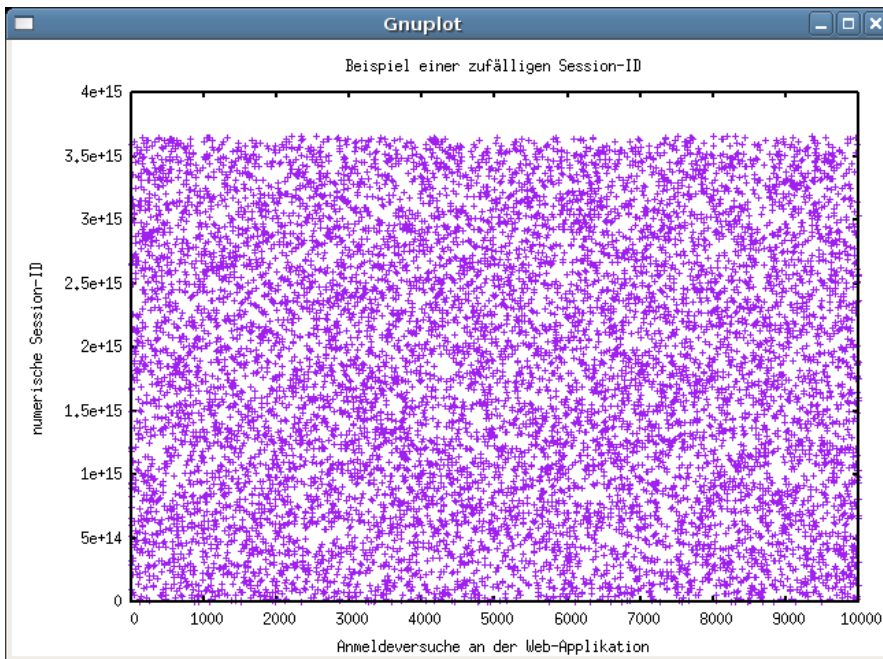


Abb. 3: Beispiel einer zufälligen Vergabe von Session-IDs anhand der graphischen Darstellung

Auch die Funktionalität des Ausloggens und des gleichzeitigen Einloggens mit ein und demselben Account („User Sharing“) wird bei der Fehlersuche analysiert.

Das Hauptaugenmerk besteht jedoch bei den Benutzereingaben. Neben den offensichtlichen Daten wie der URL und den vorhandenen Eingabefeldern (z.B. den Adressdaten) sind auch die anderen, vom Client übermittelten Daten einer Prüfung zu unterziehen. Dies sind beispielsweise

- versteckte Felder („hidden fields“), die dem Benutzer nicht angezeigt werden und häufig z.B. zur Auswahl der Sprache der Anwendung Verwendung finden
- Cookies
- die Session-ID
- HTTP-Header (z.B. die Kennungen des Web-Browsers [„User-Agent“])

Zur Prüfung werden im Wesentlichen ungültige bzw. unübliche Werte verwendet, z.B. „Karfreitag“ oder „29.02.2006“ als Datum, oder auch „'“ bzw. „<script>alert(XSS)</script>“

## 2.1 Automatisierte Werkzeuge

Um die Systemsicherheit zu prüfen, existieren viele Werkzeuge. Schwachstellen-Scanner durchsuchen Netzwerke nach Sicherheitslücken (siehe [SecScan2005]). Zahlreiche universelle Security-Scanner testen mittlerweile auch Fehler in Web-Applikationen. Der Umfang und die Detailliertheit lassen jedoch häufig zu wünschen übrig. Eine positive Ausnahme ist der in Deutschland recht unbekannt Security-Scanner Maxpatrol [7] der russischen Firma Positive Technologies, der neben allgemeinen, netzwerkbasierten Schwachstellen zahlreiche Fehler in Web-Applikationen findet.

Um Sicherheitsschwächen in Web-Applikationen zu finden, gibt es mittlerweile auch einige darauf spezialisierte Schwachstellen-Scanner (siehe [Web2006]), z.B.

- ◆ Paros [8]
- ◆ Burp [9]
- ◆ Acunetix Web Vulnerability Scanner [10]
- ◆ AppScan [11]

Wie jedoch bei Schwachstellen-Scannern üblich, kann die menschliche Kreativität nicht in Software abgebildet werden.

## 2.2 Manuelles Kombinieren

Unter Zuhilfenahme automatisierbarer Werkzeuge und menschlicher Kreativität lassen sich Kombinationen von Fehlern erzeugen. So sind häufig webbasierte Kontaktformulare anfällig für E-Mail-Angriffe, da von der Web-Applikation E-Mails generiert werden, die von einem E-Mail-Programm verarbeitet werden (und dies u.U. angreifen). Derartige komplexe Vorgänge kann eine Software nicht analysieren. Auch das erfolgreiche Ausnutzen von SQL-Injection-Schwächen setzt die Kombination vom Wissen über die Datenbank als auch das zugrundeliegende Datenbanksystem voraus. Stets wird dabei mit manipulierten Benutzereingaben gearbeitet.

## 3 Vermeidung und Behebung von Fehlern

Die folgenden Hinweise – aus der Sicht eines Angreifers – werden aufgrund der Analyse zahlreicher Web-Applikationen empfohlen.

Primär sollten sämtliche Benutzereingaben auf unerwünschte Eingaben geprüft und gefiltert werden. Zur Vermeidung von SQL-Injections sollte das Hochkomma („'“) nicht in Benutzereingaben zugelassen werden. Dies kann jedoch nicht immer vermieden werden, z.B. bei manchen Namen („O'Neill“). Deshalb sollten derartige Zeichen serverseitig HTML-kodiert gespeichert werden (im Beispiel durch „&#39;“). Die serverseitigen Filter sollten dabei alle Kodierungsformen auswerten können (Unicode in HTML Dezimal „&#39“, Unicode in HTML Hexadezimal „&#x27“, Unicode-Darstellung „%27“). Ferner sollten die Filter nicht durch führende Nullen (z.B. „&#x0027“) umgangen werden können. Zu beachten ist weiterhin, dass HTML- und Javascript meist case-insensitiv sind, d.h. Groß- und Kleinbuchstaben werden nicht beachtet. Das Unterdrücken von Eingaben wie „<script>“ ist eine prinzipiell sinnvolle Maßnahme gegen Cross-Site-Scripting. Doch „<SCRIPT>“ kann identisch zum Einschleusen schadhafte Codes verwendet werden. Auch lassen sich mitunter die HTML-eigenen Zeichen durch eine HTML-Darstellung einschleusen („&lt;ScRiPt&gt;“). Das Filtern der

Benutzereingaben darf sich serverseitig keinesfalls auf die erwarteten Benutzerdaten beschränken, sondern sollte für alle Daten – die Clients zum Server senden – durchgeführt werden (z.B. das „User-Agent“ Feld des HTTP-Headers sowie sämtliche „hidden fields“).

Die Prüfung der Benutzereingaben muss serverseitig erfolgen. Eine Prüfung, z.B. mittels JavaScript, kann mit entsprechenden Plugins leicht umgangen werden, für Mozilla z.B. mit der „Web Developer Extension“ [12] oder mittels der „HTML Source Code Explorer Bar“ [13] im Internet Explorer.

Beim Design einer Anwendung ist auf die vollständige serverseitige Implementierung aller Funktionen zu achten. Oft werden z.B. bei der Funktionalität „Logout“ lediglich die Cookies im Web-Browser gelöscht.

Die Basic Authentifizierung von HTTP sollte niemals verwendet werden, da diese generell unsicher ist [RFC1945]. Auch bei Verwendung von HTTPS weist diese Authentifizierungsart zahlreiche Schwachpunkte auf. Bei jedem Datenpaket (auch bei Bildern etc.) werden die Authentifizierungsdaten zum Server gesendet (Benutzername und Kennwort), es ist kein Abmelden möglich, da Web-Browser die Anmelde-Informationen bis zum Beenden des Web-Browsers im Arbeitsspeicher vorhalten, und auch gegen „User Sharing“ bietet diese Authentifizierungsart keinerlei Schutz.

Die Anzahl der Parameter einer Web-Applikation ist so gering wie möglich zu halten, da jede Benutzereingabe eine prinzipielle Schwachstelle ist. Im Idealfall nutzt eine Web-Applikation lediglich eine zufällige Session-ID, und jede Ansicht der Anwendung wird serverseitig aufbereitet, wobei für jede Funktionalität zufällig gewählte Funktionsnamen Verwendung finden. Die Session-ID sollte nicht in der URL, sondern als Cookie übertragen werden, da URLs an vielen Stellen gespeichert werden, z.B. im „Verlauf“ des Browsers oder bei einem Proxy.

Auch der Datenschutz ist beim Design von Web-Applikationen zu beachten. Die übertragenen Daten sollten dabei verschlüsselt werden, weshalb die Nutzung von SSL/TLS (HTTPS) nahe gelegt wird. Die Verschlüsselung der Daten ist jedoch ausschließlich aufgrund des Datenschutzes sinnvoll, denn sämtliche Fehler in Web-Applikationen lassen sich identisch via HTTPS ausnutzen. Die serverseitige Speicherung der Daten ist ebenfalls unter Aspekten des Datenschutzes zu entwerfen

bzw. zu überarbeiten. Wenn Funktionalitäten wie „Kennwort vergessen“ das Kennwort eines Benutzers an dessen hinterlegte E-Mail-Adresse im Klartext senden, so wird das Kennwort offensichtlich unverschlüsselt auf dem System abgelegt. Falls beispielsweise durch eine SQL-Injection die entsprechende Tabelle der Datenbank ausgelesen werden kann, so wären einem Dritten sämtliche Passwörter bekannt.

Die Verwendung von E-Mails in Zusammenhang mit Web-Applikationen sollte sehr detailliert betrachtet werden. Oft werden aufgrund webbasierter Anfrageformulare von Web-Applikationen E-Mails versandt, entweder an die zu erreichenden Personen, den Absender selbst (z.B. als Bestellbestätigung) oder gar an Dritte („Empfehlungen“). Nicht nur bei der letztgenannten Funktionalität besteht ein Problem bzgl. dem Versenden unerwünschter E-Mails (Spam). In allen Fällen kann es möglich sein, dass durch manipulierte Benutzereingaben die E-Mails umadressiert werden. Auch das Einschleusen von „malicious code“ ist den Autoren bei mehreren Web-Applikationen möglich gewesen. Für den Mailserver, der die E-Mails der Web-Applikation versendet, sollten die E-Mails prinzipiell als aus einer nicht vertrauenswürdigen Quelle stammend gekennzeichnet und dementsprechend verarbeitet werden. Dies beinhaltet dementsprechend sowohl Validierungen bzgl. des Inhalts („Anti-Viren“ und „Anti-Spam-Lösungen“), als auch andere Auffälligkeiten (Anzahl der E-Mails, Anzahl der Empfänger etc.).

## Fazit

Ein sicheres Netzwerk kann durch eine einzige Web-Applikation kompromittiert werden. Neben der Sicherheit der Netzwerkinfrastruktur ist sie auch für die verwendete Individualsoftware unbedingt zu gewährleisten.

- „Unvalidated input“ ist die größte Schwachstelle bei Web-Applikationen.
- Vollständige serverseitige Anwendungen sollen entwickelt werden.
- Datenschutzaspekte sind bei Web-Applikationen unbedingt zu beachten.

## Links

[1] <http://www.phpbb.com>

[2] <http://www.wireshark.org>

[3]

[http://seclists.org/webappsec/2003/q1/att-0271/charset\\_pl](http://seclists.org/webappsec/2003/q1/att-0271/charset_pl)

[4] <http://livehttpheaders.mozdev.org>

[5] <http://www.gnuplot.info>

[6] <http://www.stunnel.org>

[7] <http://www.maxpatrol.com>

[8] <http://www.parosproxy.org>

[9] <http://www.portswigger.net>

[10] <http://www.acunetix.com>

[11] <http://www.watchfire.com>

[12]

<http://chrispederick.com/work/webdevelop/>

[13]

<http://www.vdberg.org/~richard/htmlbar.html>

## Literatur

- [SQL2001] *Christoph Wille*: SQL Injection (30.10.2001)  
<http://www.aspheute.com/artikel/20011030.htm>
- [SQL2002] *Kevin Spett*: SQL Injection (Atlanta 2002)  
<http://www.spidynamics.com/support/whitepapers/WhitepaperSQLInjection.pdf>
- [CCC2004] *Dirk Heringhaus*: No more secrets? T-Com machts möglich! (Datenschleuder #83, Juli 2004, S. 16)
- [SecScan2005] *Ute Roos, Christoph Puppe*: Löchersuche: All-Purpose-Schwachstellen-Scanner im Test (iX 9/2005, S. 84)
- [Gartner2006] *Gartner RAS Core Research Note G00136839*: Management Update: Predicts 2006: Collaboration Comes of Age (07.12.2005)
- [Web2006] *Katrin Heinrich*: Sicherheitstests für Web-Applikationen, Diplomarbeit, Hochschule Reutlingen (20.01.2006)
- [Paypal2006] *Paul Mutton*: PayPal Security Flaw allows Identity Theft (16.06.2006)  
[http://news.netcraft.com/archives/2006/06/16/paypal\\_security\\_flaw\\_allows\\_identity\\_theft.html](http://news.netcraft.com/archives/2006/06/16/paypal_security_flaw_allows_identity_theft.html)
- [RFC1945] *Tim Berners-Lee*: Hypertext Transfer Protocol – HTTP/1.0 (Mai 1996)
- [TopTen] *OWASP*: Top Ten security issues of web applications  
[http://www.owasp.org/index.php/OWASP\\_Top\\_Ten\\_Project](http://www.owasp.org/index.php/OWASP_Top_Ten_Project)