Fraunhofer Institut
Experimentelles
Software Engineering

Typische Sicherheitsschwachstellen in Web-Anwendungen

Autor:
Dr. Holger Peine

IESE-Bericht Nr. 088.04/D
Version 1.1
12. November 2004

Eine Publikation des Fraunhofer IESE

Das Fraunhofer IESE ist ein Institut der Fraunhofer-Gesellschaft.
Das Institut transferiert innovative Software-Entwicklungstechniken, -Methoden und -Werkzeuge in die industrielle Praxis. Es hilft Unternehmen, bedarfsgerechte Software-Kompetenzen aufzubauen und eine wettbewerbsfähige Marktposition zu erlangen.

Das Fraunhofer IESE steht unter der Leitung von
Prof. Dr. Dieter Rombach (geschäftsführend)
Prof. Dr. Peter Liggesmeyer
Sauerwiesen 6
67661 Kaiserslautern

Inhaltsverzeichnis

1 Was sind Web-Anwendungen, und warum sind sie besonders gefährdet?

Web-Anwendungen werden heute für fast alle denkbaren über ein Computernetz angebotenen Anwendungen eingesetzt, auch für hoch sicherheitsempfindliche Anwendungen. Es sei kurz in Erinnerung gerufen, welche Auswirkungen ein erfolgreicher Angriff auf eine Web-Anwendung haben kann:

- Ausspähen oder Verfälschen von kompletten Datenbanken
- Zugriff auf Webserver und Application Server mit vollen Systemrechten
- Ausführen von Funktionen unter falscher Benutzeridentität
- Rufschädigende Verunstaltung der Website
- Nutzung des angegriffenen Rechners für weitere Angriffe auf Dritte

Allen Web-Anwendungen ist gemeinsam, dass sie Dienste über das HTTP-Protokoll zur Verfügung stellen ("http:" oder "https:"). Die eigentliche Funktion des Dienstes (z.B. eine online-Datenbank) liegt deshalb "oberhalb" der Ebene, auf der sich traditionelle Netzwerkprotokolle bewegen ("TCP/IP") und auf der auch die typischen Maßnahmen der Netzwerksicherheit wie Firewalls operieren. Web-Anwendungen können daher nicht durch solche Maßnahmen geschützt werden. Entsprechende Maßnahmen für die Web-Ebene ("Application Level Firewalls" o.ä.) stehen noch ziemlich am Anfang ihrer Entwicklung; das Hauptaugenmerk bei der Computersicherheit liegt immer noch zu sehr auf der reinen Netzwerksicherheit. HTTP-Anfragen werden weder von Firewalls, noch vom Betriebssystem oder der Server-Software geprüft. Nach dem heutigen Stand der Technik muss sich eine Web-Anwendung daher im wesentlichen "selbst", d.h. durch umsichtige Programmierung, vor Angreifern schützen.

Neben diesem fast ganz fehlenden Schutz durch Netzwerksicherheitsmaßnahmen sind Web-Anwendungen zusätzlich noch dadurch gefährdet, dass sie unabhängig von ihrer konkreten Funktion auf einem erheblichen Vorrat an gemeinsamen Techniken aufbauen (z.B. zur Sitzungsverwaltung oder zum Datenbankzugriff), die allesamt Sicherheitsschwachstellen einschleppen können, wenn sie nicht mit Sorgfalt implementiert werden. Solche typischen Schwachstellen von Web-Anwendungen sind Angreifern gut bekannt; teilweise sind sogar Werkzeuge im Internet verfügbar, um bestimmte Schwachstellen automatisch zu suchen und auszunutzen.

Der Entwickler einer Web-Anwendung sollte daher die typischen Sicherheitsschwachstellen dieser Technik kennen und vermeiden. Dieser Bericht gibt einen Überblick über solche Schwachstellen und wie man sie vermeiden kann; dabei beschränken wir uns auf die Entwicklung von *serverseitigen Anwendungen*. Dieser Bericht kann keine vollständige Einführung in die erwähnten Techniken bieten — dafür gibt es anderes, weitaus umfangreicheres Material (auf das in Abschnitt B am Ende dieses Berichts hingewiesen werden wird). Das Ziel dieses Berichts ist vielmehr, dass der Entwickler

- ein Bewusstsein für die Probleme bekommt,
- erkennen kann, welche davon in seiner Anwendung vorkommen könnten, und
- die Probleme in seiner konkreten Implementierung finden und (in der Regel mit Hilfe von weiterem Material) beheben kann.

Die technische Grenze des Begriffs "Web-Anwendung" ist nicht klar definiert; oft werden auch Techniken wie JSP/Servlets, ASP, PHP, CGI, Perl, SSI, u.v.m. dazu gezählt. Auf diese Techniken, die jeweils auch ihre speziellen Sicherheitsprobleme mitbringen, wird hier nicht eingegangen (mit Ausnahme von PHP in Abschnitt 10). Viele der hier beschriebenen Schwachstellen, z.B. das Einschmuggeln von Code, tauchen aber auch bei diesen Techniken immer wieder auf.

2 Vom Browser übermittelte Daten

Unter Sicherheitsgesichtspunkten besteht ein grundsätzlicher Unterschied zwischen Server und Browser: Das Verhalten des Servers hat der Entwickler unter Kontrolle, das Verhalten des Browsers *einschließlich aller auf Browser-Seite gespeicherten Daten und dort ablaufenden Codes wie Javascript, Java Applets etc. hat der Entwickler nicht unter Kontrolle!* Der Browser-Benutzer kann bei entsprechenden technischen Kenntnissen den Browser-Teil einer Anwendung beliebig ausspähen und manipulieren. Das hat weit reichende Konsequenzen für den Server-Entwickler, wie die folgenden Abschnitte ausführen. Alle vom Browser bzw. vom Benutzer kommenden Daten müssen daher immer auf Server-Seite erst auf Plausibilität und Zulässigkeit geprüft werden, bevor sie weiter verarbeitet werden dürfen. Ungenügende Zulässigkeitsprüfung von Eingabedaten

ist die bei weitem wichtigste Ursache von Sicherheitsproblemen bei Web-Anwendungen.

Zulässigkeitsprüfungen von Daten sollten immer so implementiert werden, dass der zu prüfende Wert mit einer Menge (bzw. einem Muster o.ä.) von *zulässigen* Werten verglichen wird — *nicht* mit einer Menge von unzulässigen ("verdächtigen", "gefährlichen") Werten. Dies deshalb, weil es schwerer ist, zu überblicken, welche Arten von Parametern in welchen nachgelagerten Anwendungen Probleme verursachen könnten (zu leicht wird hier nämlich eine Sorte von problematischen Werten übersehen), als anzugeben, welche Werte an dieser Stelle in der Anwendung sinnvoll sind.

Alle im folgenden beschriebenen Schwachstellen und Angriffe mögen in der auf das wesentliche reduzierten Darstellung hier folgerichtig oder gar offensichtlich erscheinen; trotzdem sind alle diese Schwachstellen in realen Web-Anwendungen im kommerziellen Einsatz tatsächlich aufgetreten.

2.1 URL-Parameter

Alle in der URL einer Anfrage an den Webserver codierten Informationen wie Parameterwerte ("http://...?name=wert") kann ein Angreifer beliebig ändern — er muss dazu nur in der URL-Zeile seines Browsers die gewünschten Parameter und Werte eingeben.

Ein Entwickler sollte sich auch nicht darauf verlassen, dass die Informationen in der URL so "undurchschaubar" codiert sind, dass kein Benutzer herausfinden wird, wie er sie zielgerichtet manipulieren kann. Falls die "undurchschaubare" Codierung auf Seiten des Browsers durchgeführt wird (etwa durch Javascript, s. Abschnitt 2.2), dann wird ein technisch versierter Benutzer die Codierung immer auflösen können.

Allgemein sollten Parameter nur zur Übermittlung von Informationen benutzt werden, die vom Benutzer eingeholt werden mussten, und nicht für Informationen, die auch der Server selbst im Zusammenhang mit einem Benutzer speichern könnte (s. Abschnitt 3 "Eingeschmuggelter Code"). Jeder zusätzliche Parameter kann potentiell missbraucht werden, deshalb sollten so wenig Parameter wie möglich verwendet werden.

Parameter, die durch HTTP-GET übermittelt werden, können in Bookmarks oder Favoriten Spuren hinterlassen, ebenso in HTTP-Referern oder in Log-Dateien auf Proxies. Das ist nicht unbedingt ein Sicherheitsproblem; trotzdem ist es vorsichtiger, Parameter statt durch HTTP-GET durch HTTP-POST zu übermitteln. HTTP-POST hat außerdem noch den Vorteil, dass die Gesamtlänge aller Parameter nicht begrenzt ist, wie es bei URLs der Fall ist.

2.2 Javascript

Javascript-Programme werden in die sichtbare Webseite "versteckt" eingebettet vom Webserver zum Browser des Benutzers übertragen. Der Browser führt dann den Javascript-Code aus, um z.B. der dargestellten Webseite ein dynamisches, auf Aktionen des Benutzers reagierendes Verhalten zu geben, z.B. Formulareingaben sofort zu überprüfen.

Im Prinzip könnten alle Funktionen, die typischerweise von Javascript übernommen werden, auch auf dem Webserver ausgeführt werden (nämlich indem jedesmal die Benutzeraktion an den Server gemeldet und vom Server eine neue Seite geschickt wird, die das Ergebnis der Aktion wiederspiegelt). In der Praxis wird man das nicht tun, da dann zu viel Kommunikation zwischen Browser und Webserver nötig wäre — die Anwendung wäre für den Benutzer unangenehm langsam. Trotzdem bleibt festzuhalten, dass rein unter funktionalen Gesichtspunkten die Verteilung der Funktionen zwischen Server und Browser (in Form von Javascript) beliebig ist.

Aus diesem Grund sollte man z.B. *"seinem" Javascript-Code niemals vertrauen*, sicherheitsrelevante Funktionen wie z.B. Passwortoperationen korrekt auszuführen — ein Angreifer könnte z.B. den heruntergeladenen Javascript-Code analysiert und die Passwortprüfung durch ein unbedingtes "OK" ersetzt haben.

2.3 Java-Applets

Für Java-Applets gilt das Gleiche wie für Javascript: Als Code, der auf dem Rechner des Benutzers abläuft und dort beliebig ausgespäht und manipuliert werden kann, darf ein Server nicht auf das korrekte Funktionieren eines Applets vertrauen. Auch die Tatsache, dass Applets nicht im Quelltext, sondern nur in Binärform zum Browser übertragen werden, schützt nur wenig vor Manipulationen; es gibt Werkzeuge, mit denen ein Angreifer den Quelltext eines Applets fast vollständig wiedergewinnen kann.

Java-Applets können auch unter Sicherheitsgesichtspunkten sinnvoll sein — allerdings nicht, um den Server vor dem Benutzer zu schützen, sondern nur, um den Benutzer und den Server gemeinsam vor Angriffen Dritter zu schützen (etwa das Abhören der Kommunikation zwischen Browser und Server, falls der Schutz durch SSL nicht ausreichend erscheint).

2.4 Ungeschützte Cookies

Cookies sind kleine Datenobjekte, die der Webserver dem Browser zur dauerhaften Speicherung auf der Festplatte des Benutzerrechners schickt. Bei späteren Zugriffen auf den gleichen Webserver schickt der Browser dann das einmal

erhaltene Cookie mit. Damit kann sich z.B. der Browser gegenüber dem Server als der "gleiche" Browser identifizieren, der z.B. den Zugriff auf die Seite XY vor zwei Minuten zuvor ausgeführt hatte (s. Abschnitt 5 "Schutz von Benutzersitzungen").

Ebenso wie bei Javascript oder Java Applets darf der Entwickler einer Server-Anwendung nicht darauf vertrauen, dass die Cookies, die ein Browser sendet, vor dem Endbenutzer verborgen bleiben und nicht von ihm manipuliert worden sind: Ein technisch versierter Benutzer kann alle Cookies auf seinem Rechner analysieren und beliebig verändern. Keinesfalls dürfen deshalb die in einem Cookie gespeicherten Daten ungeprüft verarbeitet werden. Einen solchen Fehler hat einmal ein online-Shop begangen, der die laufende Summe aller Waren im virtuellen "Einkaufswagen" des Benutzers in einem Cookie gespeichert hatte — ein cleverer Angreifer ersetzte einfach diese Summe durch Null und konnte so kostenfrei einkaufen.

Je nach Verwendungszweck sollten die Cookie-Daten daher geschützt werden: Falls das Cookie nur die Benutzeridentität codiert, muss es nur gegen Dritte geschützt werden, die versuchen könnten, das Cookie zu erraten, um sich so als der eigentliche Benutzer auszugeben. Für solche Fälle genügt als Schutz, im Cookie eine große Zufallszahl (mindestens 32 Bit) zu speichern, die ein Angreifer nicht erraten kann. Müssen die in einem Cookie gespeicherten Daten aber auch vor Manipulation durch den Benutzer geschützt werden (wie in dem Einkaufswagenbeispiel von oben), dann sollten sie durch den Webserver digital unterschrieben werden, bevor sie an den Browser verschickt werden. Auf diese Weise kann ein Angreifer weder ein echtes Cookie ändern, noch selbst ein gefälschtes Cookie herstellen. Falls die Daten im Cookie dem Benutzer ganz verborgen bleiben sollen, muss das Cookie neben der Unterschrift auch noch verschlüsselt werden. Wird dann ein Cookie von einem Browser an den Server übermittelt, wird erst die Unterschrift des Servers geprüft und das Cookie ggf. entschlüsselt, bevor es ausgewertet wird.

2.5 Versteckte Felder

Formulare auf Webseiten können neben den vom Browser normalerweise angezeigten Felder auch "versteckte" Felder ("<input ... type=hidden ...>") enthalten, deren Inhalte ebenfalls an den Server übermittelt werden. Solche versteckten Felder sind allerdings ungeeignet, um Daten zu übermitteln, die vor dem Benutzer verborgen werden sollen — es braucht nur geringe technische Kenntnisse, um die "versteckten" Felder zu entdecken: Einfach den HTLML-Quelltext der Seite anschauen und die "versteckten" Felder mit einem Text-Editor ändern.

2.6 Überlange Eingabedaten

Eingabedaten vom (böswilligen) Benutzer können auch ganz unabhängig von ihrem Inhalt allein durch ihre Länge Fehlfunktionen des Servers auslösen und Angriffswege eröffnen. Der Grund liegt darin, dass Software an vielen Stellen — ob im Webserver, Application Server, Betriebssystem, in der Datenbank oder in irgendwelchen Anwendungen — stillschweigend davon ausgeht, dass bestimmte Daten gewisse Maximalgrößen nicht überschreiten. Ist dies aber doch der Fall und wird nicht bemerkt, dann können überlange Daten interne Speicherbereiche solcher nachlässig programmierter Software überschreiben und damit beliebige Fehlfunktionen auslösen, die sich bei geschickter Wahl der Daten auch gezielt ausnutzen lassen.

Solche "Pufferüberläufe" treten zwar nur in Software auf, die in der Programmiersprache C implementiert wurde, aber solche Software ist in Form von Bibliotheken sehr weit verbreitet, ohne dass dies dem Anwendungsprogrammierer bewusst ist (z.B. programmiert er vielleicht in der Sprache Perl, deren Standardbibliothek aber in C implementiert ist).

Zum Schutz vor Pufferüberläufen sollten alle Eingabedaten vor der Weiterverarbeitung auf vernünftige Längen abgeschnitten werden. Was dabei jeweils "vernünftig" ist, hängt natürlich stark vom jeweiligen Verwendungszweck der Daten ab.

2.7 HTTP_REFERER

Das HTTP_REFERER-Feld in einer HTTP-Anforderung gibt an, welcher Seite der anfordernde Browser zuletzt besucht hatte. Diese Information kann ein technisch versierter Benutzer leicht fälschen. Ein Webserver darf daher dem Wert dieses Felds nicht trauen.

3 Eingeschmuggelter Code

Will ein Angreifer eine Web-Anwendung dazu bringen, bestimmte Funktionen auszuführen, die sie normalerweise nicht ausführt, dann er muss er der Anwendung den Code für diese Funktionen auf irgendeine Weise "unterschieben". Im

Prinzip kann dies Code für jedwede "hinter" bzw. "unter" dem Webserver liegende Software sein, die Code zur Ausführung akzeptiert; die häufigsten Fälle sind SQL-Code für eine hinter dem Webserver liegende Datenbank, und Shell-Code, d.h. Kommandos, für das "unter" dem Webserver liegende Betriebssystem. Aber auch Software wie LDAP oder Sendmail akzeptiert ausführbaren Code, der zu Angriffen missbraucht werden kann.

Angriffscode wird meistens als Benutzereingabe in ein Formular geschrieben, dessen Inhalt dann auf dem Webserver verarbeitet wird. Die erste Verteidigungslinie gegen eingeschmuggelten Code ist daher die genaue Prüfung aller Benutzereingaben wie am Anfang von Abschnitt 2 beschrieben, denn Code stellt so gut wie nie eine sinnvolle Benutzereingabe dar. Der Angriffscode kann auch im "%nn"-Format von URLs oder im "&#nn;"-Format von HTML-Entities codiert sein, um seine Entdeckung zu erschweren. Deshalb sollten die zu prüfenden Daten immer vor der Prüfung decodiert werden.

Allgemein sollte möglichst wenig Code dynamisch aus Benutzereingaben erzeugt werden, und stattdessen möglichst viel Code fertig bereit liegen. Dies gilt besonders für Code zum Datenbankzugriff; hier sollten möglichst "Prepared Statements" verwendet werden.

4 Benutzeranmeldung und Passwörter

Benutzer einer sicherheitsempfindlichen Web-Anwendung müssen sich in der Regel anmelden und ihre Identität nachweisen ("authentisieren"), um alle Funktionen der Anwendung nutzen zu können. Abgesehen von hardwarebasierter Authentisierung (z.B. über SmartCards oder biometrische Geräte) geschieht dies fast immer durch die Eingabe eines Benutzernamens und Passworts. Die Handhabung von Passwörtern ist daher von zentraler Bedeutung für die Sicherheit einer Web-Anwendung.

Eingegebene Passwörter sollten nie über eine ungesicherte HTTP-Verbindung verschickt werden, sondern nur über eine SSL-Verbindung.

Passwörter sollten möglichst nicht im Klartext gespeichert werden, sondern nur als Hash. So kann ein Angreifer mit den (gehashten) Passwörtern nichts anfangen, falls sie ihm in die Hände fallen sollten.

Falls ein Benutzer sein Passwort vergessen hat, wird ihm manchmal die Möglichkeit angeboten, das Passwort per E-Mail an eine zuvor vom Benutzer angegebene Adresse zu verschicken. Nach einer solchen Aktion sollten hoch sicherheitsempfindliche Daten wie z.B. Kreditkartennummern vom Server gelöscht werden, so dass sie der Benutzer erneut eingeben muss. Außerdem setzt diese Methode voraus, dass das Passwort irgendwo auf dem Server im Klartext gespeichert sein muss, was sich nicht mit der Empfehlung verträgt, Passwörter nur als Hash zu speichern. In diesem Fall muss das Passwort zumindest verschlüsselt werden; der Schlüssel dazu sollte nicht in einer Datei auf dem Webserver gespeichert werden, sondern dem Webserver als Aufrufparameter o.ä. bei jedem Start mitgegeben werden.

5 Schutz von Benutzersitzungen

Wenn ein und derselbe Benutzer verschiedene Seiten von einem Webserver anfordert (etwa weil er verschiedene Eingaben unter seinem Namen tätigen möchte, z.B. verschiedene Waren in seinen Einkaufswagen bei einem online-Shop legen will), dann kann der Webserver zunächst einmal nicht erkennen, dass diese Zugriffe alle demselben Benutzer zuzuordnen sind; rein aus dem HTTP-Protokoll ist nämlich nicht ersichtlich, von welchem Browser ein HTTP-Zugriff kommt (HTTP ist ein “zustandsloses Protokoll”). Deshalb implementieren praktisch alle Web-Anwendungen, die eine Benutzeranmeldung kennen, auch sogenannte Benutzersitzungen. Eine Sitzung (“Session”) umfasst alle Zugriffe von ein und demselben Benutzer von seiner Anmeldung bis zu seiner Abmeldung. Da HTTP keine Sitzungen “eingebaut hat”, muss der Browser die Sitzung durch die Angabe bestimmter bei der Anmeldung erhaltener Daten bei jedem Zugriff dem Webserver mitteilen.

Für solche Sitzungsidentifikationsdaten werden oft Cookies verwendet (“Session cookies”). Solche Cookies müssen geschützt werden wie in Abschnitt 2.4 beschrieben. Als Schutz gegen den Diebstahl von Session-Cookies (s. z.B. Abschnitt 7 "Cross-Site Scripting") empfiehlt es sich, die IP-Adresse des Benutzerrechners für jede Session für die Dauer der Session auf dem Server zu speichern.

Sitzungen sollten immer durch SSL geschützt werden, um eine illegale Übernahme durch Dritte ("Session hijacking") z.B. durch Ausspähen und eigene Verwendung der Sitzungsidentifikationsdaten zu erschweren.

Die zu einer Sitzung gehörigen Informationen (z.B. Beginn der Sitzung, IP-Adresse des Benutzers, zuletzt erfolgter Schritt innerhalb einer vorgegebenen Abfolge von Schritten in der Anwendung etc.) sollten auf dem Server gespeichert werden, nicht etwa in Daten, die der Browser übermittelt (zu den Gründen s. Abschnitt 2). Falls die Sitzungsinformationen nicht auf dem Server gespeichert werden können (z.B. weil sehr viele und sehr lange dauernde Sitzungen zu viel Speicherplatz auf dem Server benötigen würden), können Sitzungsinformationen auch in Cookies gespeichert werden, vorausgesetzt, die Cookies werden geschützt wie in Abschnitt 2.4 beschrieben.

Sitzungen sollten nach einer bestimmten Zeit ohne Benutzer-Zugriffe vom Server automatisch beendet werden; das senkt die Gefahr, dass "herrenlose" Sitzungen von Angreifern übernommen werden. Ebenso sollten lang laufende Sitzungen unabhängig von der Benutzeraktivität periodisch eine Neuanmeldung des Benutzers verlangen; dies beschränkt die Zeit, in der ein Angreifer eine wie auch immer "eroberte" Sitzung nutzen kann.

6 Zugriffskontrolle

Im allgemeinen soll nicht jeder Benutzer einer Anwendung alle Daten der Anwendung lesen oder schreiben können. Als allererste Maßnahme, um dies sicher zu stellen, sollte explizit festgelegt werden, welche Benutzer(-gruppen) welche Daten lesen, anlegen, ändern, oder löschen dürfen. Diese "Sicherheitspolitik" ("Security Policy") sollte Teil der Anforderungsbeschreibung für die Web-Anwendung sein.

Um diese Politik in der Implementierung auch umzusetzen, steht eine Vielzahl von Möglichkeiten zur Verfügung, so dass allgemeine Empfehlungen schwer fallen — Application Server, Datenbank, und Betriebssystem bieten jeweils eigene Mittel zur Zugriffskontrolle an, einschließlich eines jeweils eigenen Benutzerkontos. Daneben gibt es noch separate Produkte zur Zugangskontrolle. Entscheidend ist, dass jeder Punkt der Sicherheitspolitik durch eine explizite Maßnahme (z.B. ein Dateizugriffsrecht auf Betriebssystemebene) abgedeckt ist, und dass

auch explizit dokumentiert wird, durch welche Maßnahme jeder Punkt abgedeckt ist.

7 Cross-Site Scripting

Cross-Site Scripting (XSS) ist ein Angriff, bei dem es drei Beteiligte gibt: Einen Angreifer, eine missbrauchte Website, und das eigentliche Opfer. Der Angreifer sorgt dafür, dass das Opfer die Website besucht (z.B. durch einen Link in einer E-Mail an das Opfer), und dass die Website dem Opfer einen vom Angreifer stammenden Angriffscode in einer Webseite versteckt liefert. Die Schwachstelle der Website, die hier missbraucht wird, besteht darin, dass die Website von fremden Benutzern Code akzeptiert und als Antwort auf Anfragen anderer Benutzer diesen Code auch wieder ausgibt.

Zum Missbrauch für XSS-Angriffe eignen sich grundsätzlich alle Websites, die von Benutzern auf einer Webseite Eingabedaten akzeptieren, die später in die Ausgabe anderer Webseiten eingebaut werden. Das können z.B. Seiten sein, die zur Ablage von Benutzerkommentaren gedacht sind (Foren, Gästebücher), aber auch Sucheingabemasken, die das eingegebene Suchmuster in der Seite mit den Suchergebnissen wiederholen ("Ihre Suche nach 'XY' lieferte folgende Treffer: ..."). Im letzten Fall muss der Angriffscode nicht einmal auf der Website gespeichert werden — aus der Suchanfrage wird ohne Speicherung direkt die Ergebnisseite erzeugt. Der Angreifer schmuggelt also den Angriffscode ein, indem er ihn als Benutzerkommentar oder Suchmuster in die verwundbare Seite eingibt. Andere Benutzer, die diese Seite dann aufrufen, führen den Code nichtsahnend in ihrem Browser aus. Der für XSS benutzte Code ist i.a. Javascript-Code; prinzipiell ist aber jeder von einem Browser ausführbare Code geeignet (z.B. etwa Flash oder ActiveX).

Da die Möglichkeiten zum Zugriff auf das lokale System für im Browser ablaufenden Code beschränkt sind (Javascript hat z.B. keinen freien Zugriff auf die Festplatte des Opfers), besteht das Ziel eines XSS-Angriffs in der Praxis meist darin, das Session-Cookie des Opfers aus seiner Sitzung mit der missbrauchten Website (s. Abschnitt 5) auszuspähen: Der Angriffscode sendet dieses Cookie vom Rechner des Opfers zu dem des Angreifers, der das Cookie dann benutzen kann, um die Sitzung zu "entführen" ("session hijacking"), also ohne weitere

Anmeldung unter der Identität des Opfers mit der missbrauchten Website zu arbeiten.

Schutz vor dem Missbrauch einer Web-Anwendung für XSS-Angriffe bietet nur eine gründliche Prüfung aller vom Benutzer (hier: Angreifer) eingegebenen Daten, wie in Abschnitt 2 "Vom Browser übermittelte Daten" beschrieben. Ausführbarer Code wie Javascript sollte durch diese Prüfung grundsätzlich ausgeschlossen werden (es sei hier aber nochmals darauf hingewiesen, dass Gültigkeitsprüfungen grundsätzlich auf Zulässigkeit prüfen sollten, nicht auf Unzulässigkeit!). Falls ausführbarer Code tatsächlich als sinnvoller Inhalt der Website vorkommen kann (z.B. in einem Forum zum Thema Javascript-Programmierung), dann muss er so codiert werden (z.B. als HTML-Entities), dass er vom Browser zwar dargestellt, aber nicht als Code ausgeführt wird.

8 Verräterische Fehlermeldungen

Ein Angreifer wird versuchen, die Anwendung zu Fehlern zu provozieren, um aus ihrem Fehlerverhalten Rückschlüsse auf ihre interne Struktur zu ziehen. Fehlermeldungen sollten daher den Fehler ausschließlich aus Benutzersicht beschreiben und keine Informationen über die interne Struktur der Web-Anwendung offenbaren. Fehlermeldungen sollten z.B. keine Codestücke, Stack-Traces, Klassennamen, Funktionsnamen, Tabellennamen, Hinweise auf "Baustellen" im Code, oder Entwicklernamen enthalten — alle diese Informationen könnten einem Angreifer hilfreich sein. Z.B. sollte bei Eingabe eines nicht existierenden Benutzernamens in eine Web-Anwendung keine Meldung der Art "SQL-ERROR 23477: No such row in table CORE_USERS" zurückgegeben werden (die einem Angreifer die für die weitere Angriffserkundung wertvolle Information über den Namen einer internen Datenbanktabelle offen legt), sondern eine Meldung wie "Dieser Name existiert nicht", was für einen legalen Benutzer genauso informativ ist.

Die exakte Beschreibung des Fehlers sollte stattdessen in einer normalen Benutzern nicht zugänglichen Log-Datei gesichert werden, die dann für Debugging- und Wartungszwecke die gleiche Information liefert wie eine "redselige" Fehlermeldung, ohne dabei die Sicherheit zu beeinträchtigen.

9 Verräterische Kommentare

Analog zu den Bemerkungen über Fehlermeldungen in Abschnitt 8 sollten in den an den Browser übermittelten Web-Seiten keine Kommentare (in HTML, Javascript etc.) enthalten sein, die interne Informationen offenbaren. Auch in Code, der nur auf dem Server ausgeführt wird (z.B. PHP, JSP, ASP, CGI, SSI) sollten solche Informationen möglichst vermieden werden, da es durch Fehler im Webserver passieren kann, dass Seiten mit serverseitigem Code im Quelltext, also ohne Ersetzung des Codes, an den Browser ausgeliefert wird.

10 PHP-Schwachstellen

PHP ist eine Programmiersprache für kurze Codestücke ("Skripte"), die in HTML-Seiten eingebettet werden. Die Skripte werden allerdings nicht mit dem Rest der Seite zum Browser übertragen, sondern vom Webserver vor der Übertragung ausgeführt. Damit erhält die Seite eine dynamische Gestalt: Je nach dem, wie sich das PHP-Skript verhält, fällt der HTML-Code, der zum Browser geschickt wird (und damit das Aussehen der Webseite), jedesmal anders aus.

PHP-Skripte werden vor allem dazu eingesetzt, Webseiten in Abhängigkeit von Benutzereingaben zu gestalten. Deshalb bietet PHP auch zahlreiche Mittel, um Benutzereingabedaten wie URL-Parameter, Cookies, HTTP-Header-Felder, hochgeladene Dateien etc. möglichst nahtlos in das PHP-Skript einzufügen. Genau dieser Komfort von PHP birgt aber auch ein großes Gefahrenpotential, da alle diese Eingabedaten für Angriffe genutzt werden können. PHP eröffnet deshalb zahlreiche Möglichkeiten für Angriffe. Viele davon können durch Verzicht auf bestimmte Komfortmerkmale von PHP (bei der Programmierung oder bei der Konfiguration des PHP-Systems) verhindert werden, wie im folgenden dargestellt wird.

Dieser Abschnitt basiert in wesentlichen Teilen auf [Scarlet], wo noch weitere Details und Beispiele zu finden sind.

10.1 Extern schreibbare Variablen

In der voreingestellten Konfiguration von PHP erzeugen alle vom Browser-Benutzer über HTTP gesetzten Variablen (z.B. URL-Parameter, Cookies, HTTP-Header-Felder, hochgeladene Dateien) automatisch eine gleichnamige, globale Variable mit dem vom Benutzer angegebenen Wert im aufgerufenen PHP-Skript. Dieses Verhalten von PHP, so komfortabel es für den Programmierer auch ist, gibt einem Angreifer tiefen Eingriff in das Verhalten des Skripts und damit zahlreiche Angriffspunkte, denn ein Angreifer kann bei etwas Kenntnis des PHP-Quelltexts (was oft nicht schwer zu erlangen ist, z.B. durch Probieren) auch anwendungsinterne Variablen manipulieren. Ein ebenso einfaches wie desaströses Beispiel ist die Manipulation einer Variable, die das Ergebnis der Benutzerauthentisierung festhält:

```
<?php
if ($eingegebenesPasswort == "richtiges Passwort")
   $istAuthentisiert = 1;
...
if ($istAuthentisiert == 1)
    echo "Geheiminformation";
?>
```

Falls der Angreifer dieses Skript mit dem Parameter `skript.php?istAuthentisiert=1` aufruft, wird er unabhängig vom eingegebenen Passwort als authentisiert behandelt.

Gegen solche Angriffe kann sich der PHP-Programmierer schützen, indem er jede Variable, die nicht für Benutzereingaben gedacht ist, vor der ersten Verwendung löscht (z.B. `$var = ""`).

Bei PHP-Anwendungen, die aus mehreren ineinander verschachtelten Skripten bestehen, die über gemeinsam genutzte globale Variablen miteinander kommunizieren, besteht hier eine besondere Gefahr: Wenn z.B. in `a.php` eine Variable gelöscht wird und dann `b.php` von `a.php` aufgerufen wird (z.B. durch ein `include` oder `require` Kommando in `a.php`) und `b.php` sich darauf verlässt, dass `a.php` die Variable gelöscht hat, dann kann ein Angreifer `b.php` manipulieren, indem er es direkt (also nicht aus `a.php` heraus) aufruft und so doch einen manipulierten Wert für die Variable setzen kann. Es erfordert daher einige Überlegung und Disziplin, sicher zu stellen, dass wirklich immer alle globalen Variablen vor ihrer Benutzung gelöscht oder (falls sie schon sinnvolle Werte haben sollten, wie etwa in `b.php`) auf Gültigkeit geprüft werden. Zur Kommunikation zwischen Teilen einer Anwendung sollten besser andere Mittel als globale Variablen verwendet werden, z.B. Aufrufparameter.

Das automatische Setzen von Variablen durch den Aufrufer lässt sich durch das PHP-Kommando `set register_globals off` ganz ausschalten; diese Vari-

ablen sind dann nur noch durch die Arrays `HTTP_GET_VARS`, `HTTP_POST_VARS`, `HTTP_COOKIE_VARS`, und `HTTP_POST_FILES` zugänglich; interne Variablen der Anwendung kann ein Angreifer dann nicht mehr beeinflussen. Diese Maßnahme ist der sicherste Schutz, erfordert allerdings typischerweise Änderungen an zahlreichen Stellen in PHP-Skripten, die sich auf das automatische Setzen von Variablen durch den Aufrufer verlassen.

Die verbreitete Gewohnheit, *alle* in einer Anwendung benutzten Dateien (z.B. auch Konfigurationsdateien, Datendateien, Benutzertabellen etc.) formal als PHP-Skript-Dateien (erkennbar an einer Dateiendung wie `.php`) zu behandeln, macht diese Schwachstelle besonders gefährlich: Auch solche Dateien kann ein Angreifer als "PHP-Skript" mit gezielt manipulierten globalen Variablen aufrufen und so z.B. vertrauliche Daten erhalten.

10.2 Lesen externer Dateien

Viele PHP-Funktionen, die mit Dateien arbeiten (z.B. Lesen aus einer Datei) akzeptieren nicht nur die Namen lokaler Dateien, sondern auch externer Dateien aus dem Internet in Form von URLs. Offensichtlich kann ein Angreifer diese Funktion missbrauchen, um die Anwendung dazu zu bringen, Dateien mit Angriffscode vom Rechner des Angreifers zu laden.

Wo dieser Angriffscode dann platziert wird und wie er aufgerufen wird, hängt stark von der jeweiligen Anwendung ab — z.B. könnte der Angreifer durch das Setzen einer globalen Variable (s. Abschnitt 10.1), in der die Anwendung den Dateipfad zu einer Skriptbibliothek speichert, die Anwendung zum Laden und Ausführen von Skript-Code aus einer gleichnamigen Bibliothek auf dem Rechner des Angreifers veranlassen.

In jedem Fall aber ist das Lesen externer Dateien ein Einfallstor für Angriffe, dessen Größe den begrenzten legalen Nutzen dieser Funktion in fast allen Anwendungen deutlich übersteigt.

10.3 Hochladen externer Dateien

PHP lädt in seiner voreingestellten Konfiguration jede Datei vom Rechner eines externen Benutzers, die über ein HTML-Formular der Art

```
<FORM METHOD="POST">
<INPUT TYPE="FILE" NAME="dateiname">
...
```

angegeben wird, automatisch auf den Webserver hoch und speichert sie dort — mit anderen Worten, jeder Internetbenutzer kann beliebige Dateien auf den

Webserver hochladen. Solche hochgeladenen Dateien können z.B. verwendet werden, wenn Skripte ausdrücklich prüfen, ob Dateien, mit denen sie arbeiten wollen, im lokalen Dateisystem liegen, um Angriffe zu verhindern, wie sie im Abschnitt 10.2 "Lesen externer Dateien" wurden — lädt der Angreifer die Datei mit dem Angriffscode erst auf den Server hoch, dann würde sie eine solche Prüfung passieren.

Auch wenn die Funktion zum Hochladen von Dateien deaktiviert wurde, kann ein Angreifer noch einen anderen Weg benutzen, um seine Daten in eine Datei auf dem Webserver-System zu platzieren: Die Werte der Session-Variablen werden von PHP in einer Datei mit leicht zu erratendem Namen und Speicherort gespeichert. Falls die Anwendung Daten vom Benutzer übernimmt und ohne Prüfung in einer Session-Variable speichert, kann der Angreifer z.B. durch Platzierung von PHP-Code in den Eingabedaten Angriffscode hochladen. Auch hier müssen also wieder Benutzereingaben (hier: Sitzungsdaten) vor der Verarbeitung (hier: Speicherung in einer Session-Variablen) genauestens auf Zulässigkeit geprüft werden.

10.4 Maßnahmen zum Schutz von PHP-Anwendungen

Die folgenden PHP-Konfigurationsbefehle verhindern die meisten der beschriebenen Angriffe, bringen aber auch Einschränkungen der Funktion von PHP mit sich, die Änderungen in bestehenden Skripten erfordern können. Die Maßnahmen sind deshalb geordnet, von solchen mit wahrscheinlich geringen Änderungsauswirkungen, aber auch geringer Schutzwirkung, bis zu solchen mit wahrscheinlich hoher Schutzwirkung, aber auch hohen Änderungsauswirkungen:

1 `set allow_url_open off`

   Verhindert das Lesen (und jeden anderen Zugriff) von externen Dateien

2 `set display_errors off`

   Verhindert die Anzeige von Fehlermeldungen an den Benutzer (s. Abschnitt 8). Diese Anzeige ist während der Entwicklung der Anwendung sinnvoll, nicht aber für Anwendungen im Einsatz.

3 `set open_basedir`

   Begrenzt Dateioperationen auf bestimmte Verzeichnisse.

4 `set safe_mode on`

Zahlreiche Einschränkungen, z.B. bei der Menge der verfügbaren Funktionen und Kommandos, bei Dateizugriffen (je nach Zugriffsrechten von Skript und Datei), und beim Hochladen von Dateien (ganz unterbunden).

5 `set register_globals off`

Wurde in Abschnitt 10.1 beschrieben.

A Checkliste

Dieser Abschnitt fasst die meisten Ausführungen der vorigen Abschnitte als Ratschläge in knapper Form zusammen.

A.1 Allgemeines

- Zulässigkeitsprüfungen von Eingabedaten immer in positiver Form ausführen (Prüfung auf Zulässigkeit), nicht in negativer Form (Prüfung auf Unzulässigkeit).
- Alle Eingabedaten vor der Weiterverarbeitung auf vernünftige Länge beschränken.

A.2 HTTP-Request-Parameter

- Verschleierung/Codierung von Parameterwerten nicht zum Schutz sicherheitsempfindlicher Daten verwenden.
- Alle Parameterwerte auf Zulässigkeit überprüfen, dabei Codierungen wie `%nn` und `&#nn;` vor der Prüfung auflösen.
- Keine Daten im HTTP-Request transportieren, die auf Serverseite schon bekannt sind.
- Möglichst POST statt GET benutzen.
- Keine Zugriffsrechte allein auf Grund des HTTP_REFERER vergeben.

A.3 Javascript, Java-Applets

- Javascript und Java-Applets nur zur Beschleunigung und Bequemlichkeit des Benutzers verwenden (z.B. Syntaxprüfung von Benutzereingaben), nicht zur Prüfung sicherheitsrelevanter Bedingungen.

A.4 Cookies

- Falls Cookies nur zur Benutzeridentifikation eingesetzt werden: Eine große Zufallszahl (mindestens 32 Bit) im Cookie speichern.
- Falls sicherheitsempfindliche Daten im Cookie gespeichert werden oder das Cookie als Index in sicherheitsempfindliche Daten auf dem Server verwendet wird: Cookie durch den Server signieren und die Signatur vor der Verarbeitung prüfen.

A.5 Versteckte Felder

- Versteckte Felder nur zur Bequemlichkeit verwenden, nicht zur Übermittlung sicherheitsempfindlicher Daten.

A.6 Passwörter

- Passwörter niemals über einfaches HTTP übertragen, immer nur über SSL.
- Passwörter möglichst gar nicht auf dem Server speichern, sondern nur einen Hashwert des Passworts.
- Falls Passwörter auf dem Server gespeichert werden müssen, dann nur verschlüsselt, wobei der Schlüssel selbst nicht auf dem Server gespeichert wird.

A.7 Sitzungen

- Alle Daten einer Sitzung auf dem Server speichern, nicht auf dem Client oder in Cookies.
- IP-Adresse des Clients als Teil der Sitzungsdaten speichern und bei jedem Zugriff innerhalb der Sitzung prüfen.
- Sitzungen nach kurzer Zeit ohne Aktivität beenden, nach langer Zeit mit Aktivität erst nach erneuter Anmeldung fortsetzen.

A.8 Zugriffsschutz

- Für jedes sicherheitsempfindliche Datenobjekt (z.B. Dateien, Skripte) der Anwendung festlegen, welche Benutzer(gruppen) welche Rechte auf das Objekt haben, und fest halten, durch welche Maßnahmen dies sicher gestellt wird.

A.9 Cross-Site Scripting

- Für alle vom Benutzer eingegebenen Daten, die in später auszugebende Webseiten eingebaut werden, sicher stellen, dass kein ausführbarer Code (z.B. Javascript) darin enthalten ist.

A.10 Verschwiegenheit

- Keine Kommentare in an den Browser ausgelieferten Daten (z.B. HTML, Javascript) lassen.
- Keine systeminternen Informationen in Fehlermeldungen offen legen.

A.11 PHP

- Alle globalen PHP-Variablen, die nicht ausdrücklich zur Übermittlung von Benutzerdaten gedacht sind, vor der ersten Verwendung löschen (auch wenn die Verwendungen über mehrere Skripte verstreut sind).
- Auf Benutzereingaben möglichst nicht durch globale Variable zugreifen, sondern durch die Arrays HTTP_GET_VARS etc. Absolut sicher gestellt wird dies durch `set register_globals off`.
- Die Dateiendung .php nur für PHP-Skripte verwenden, nicht für Datendateien o.ä.
- Laden externer Dateien verbieten durch `set allow_url_open off`.
- Hochladen von Dateien unterbinden, wo nicht unbedingt erforderlich. Das ist möglich z.B. durch `set safe_mode on`.
- Dateioperationen auf dafür gedachte Verzeichnisse begrenzen durch `set open_basedir`.
- Anzeige von Fehlerursachen unterbinden durch `set display_errors off`.
- Möglichst `set safe_mode` benutzen.

B Literaturhinweise

[Advosys] Advosys Consulting: *Writing Secure Web Applications*
http://advosys.ca/papers/web-security.html

Konkrete Tipps in knapper Form (12 Seiten)

[OWASP Top10] The Open Web Application Security Project: *The Ten Most Critical Web Application Security Vulnerabilities*
http://www.owasp.org/documentation/topten.html

Liste von Problemen, Lösungen nur angerissen, 28 Seiten.

[OWASP Guide] The Open Web Application Security Project:
A Guide to Building Secure Web Applications
http://www.owasp.org/documentation/guide/
guide_downloads.html

Probleme und Lösungsvorschläge, 70 Seiten

[Microsoft] J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan:
Improving Web Application Security
Microsoft Press, 2003. Available in full text at
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp

Umfangreiche Darstellung mit Schwerpunkt auf Microsoft-Servern und .NET (aber vieles auch allgemein nützlich); über 900 Seiten

[Scarlet] Shaun Clowes: *A Study In Scarlet - Exploiting Common Vulnerabilities in PHP Applications*
http://www.secureareality.com.au/studyinscarlet.txt

Die zahlreichen Sicherheitsprobleme von PHP auf weniger als 10 Seiten beschrieben

Dokumenten Information

Titel:	Typische Sicherheits-Schwachstellen in Web-Anwendungen
Datum:	12. November 2004
Report:	IESE-088.04/D
Status:	Final
Verteiler:	Public