

Verwundbarkeiten von Web-Anwendungen

Christopher Holm, Christopher Schwardt

3holm@informatik.uni-hamburg.de, 3schwardt@informatik.uni-hamburg.de

Abstract: Web-Applikationen werden heute immer wichtiger, für viele Firmen sind sie die Haupteinnahmequelle, für einige auch die Einzige. Gleichzeitig werden diese Applikationen auch immer umfangreicher und komplexer. Diese Entwicklung führt dazu, dass der Code immer schwerer zu überblicken ist und auch ein *Audit* der gesamten Applikation den Zeit- und Kostenrahmen vieler Unternehmen sprengen würde. Durch die Zusammenarbeit vieler Menschen schleichen sich häufig logische Fehler in die Anwendung. Diese sollen hier jedoch nicht thematisiert werden. Wir konzentrieren uns hingegen auf gängige Implementationsschwachstellen. Häufig könnten derartige Sicherheitslücken durch eine bessere Ausbildung und Sensibilisierung der Programmierer vermieden werden.

1 Cross Site Scripting (XSS)

Eine der heutzutage am bekanntesten und mit am weitesten verbreiteten Sicherheitslücken in Web-Applikationen ist *Cross-Site Scripting*. XSS bezeichnet das Einfügen eigenen Codes, meist *JavaScript*, in fremde Seiten.

Die große Gefahr hierbei liegt darin, dass JavaScript mächtig genug ist, den gesamten Seiteninhalt einer für XSS anfälligen Seite zu verändern. Mit Hilfe von *iframes* und dem seit Web 2.0 so bekannt gewordenen *XMLHttpRequest* Objekt können sogar komplexe Aktionen im Namen des Benutzers ausgeführt werden, ohne, dass dieser etwas davon mitbekommt. Ausserdem können so *Cookies* dieser Domain ausgelesen und verändert werden. Somit kann der Angreifer das Verhalten der Seite nach Belieben ändern, Aktionen im Namen des Opfers ausführen oder sensitive Daten auslesen.

XSS ist eine der wenigen Methoden, um die *Same Origin-Policy* zu umgehen. Diese besagt, dass nur Daten, die von der gleichen Domain stammen, ausgelesen werden dürfen. Eine Subdomain darf nur auf Daten ihrer eigenen Subdomain, Subsubdomains von ihr und der explizit nur über ihr liegenden Domains zugreifen, nicht jedoch auf Daten benachbarter Subdomains.

Letztendlich sind die Möglichkeiten nur durch die Kreativität des Angreifers beschränkt. Es bleibt noch abzuwarten bis die ersten Würmer die im Moment so beliebten Online-Communities heimsuchen.

2 Session Riding/Cross Site Request Forgery (CSRF)

Cross-Site Request Forgery wurde etwa 2000/2001 bekannt und ist somit eine noch relativ neue Schwachstelle, zieht nun jedoch langsam immer mehr Aufmerksamkeit auf sich. Der neuere Name, *Session Riding*, beschreibt die Sicherheitslücke etwas plastischer, nämlich das Weiterbenutzen einer bestehenden Session.

Ziel einer solchen Attacke ist es, Aktionen auf einer Webseite durchzuführen, dies aber im Namen eines fremden Benutzers.

Das Standardszenario ist folgendes:

Der Benutzer wird auf eine Seite gelockt, die sich unter der Kontrolle des Angreifers befindet. Auf dieser wird im Hintergrund die Webseite mit gezielten Parametern geladen, auf der der Angreifer nun eine Aktion im Kontext des Opfers durchführen will. Unter der Voraussetzung, dass der Benutzer zu diesem Zeitpunkt auf dieser Seite angemeldet ist, wird sein Browser bei der Anfrage einen Cookie mitsenden, so dass die Aktion im Namen des Benutzers ausgeführt werden kann.

So können zum Beispiel, falls keine Vorkehrungen gegen solche Angriffe getroffen wurden, Passwörter oder andere Benutzereinstellungen ohne Wissen des Angegriffenen geändert oder auch Nachrichten im Namen des Benutzers verschickt werden, je nachdem, welche Services die angegriffene Seite anbietet.

Leider wird dieser Schwachstelle nicht so viel Aufmerksamkeit geschenkt und so ist eine derartige Lücke heutzutage, einfach auf Grund mangelnder Bekanntheit, verbreiteter als zum Beispiel SQL-Injection-Verwundbarkeiten (siehe unten), gegen die in Programmiersprachen teilweise bereits standardmäßig Vorkehrungen getroffen werden, wie *Magic Quotes*.

3 Session Hijacking

Beim sogenannten *Session Hijacking*, einer Angriffstechnik, die meist, aber nicht ausschließlich, auf unverschlüsselte Kommunikationsverbindungen angewendet wird, übernimmt der Angreifer mit Hilfe gültiger Session- oder Authentifizierungs-Informationen des Benutzers dessen laufende Sitzung.

Die Angriffsmöglichkeiten sind vielfältig und reichen von der Übernahme einer *HTTP*-Session (z.B. Zugriff auf den passwortgeschützten Bereich einer Webanwendung) bis hin zur Entwendung einer *TCP*-Verbindung (z.B. *Telnet*-Hijacking).

Zuerst sammelt der Angreifer mit Hilfe von *Sniffing*, *Man-in-the-middle*- oder sonstigen Angriffen ausreichend Daten, um die Session übernehmen zu können. Dann „entführt“ er sozusagen die Sitzung des Angegriffenen und führt sie selbst weiter.

Für den Angegriffenen erscheint die Übernahme meist wie jeder andere hin und wieder vorkommende Verbindungsverlust und er ist nicht in der Lage, sie als Angriff zu identifizieren.

4 Session Fixation

Session Fixation ist eine Technik, mit der man die Session eines Benutzers übernehmen kann. Sie basiert darauf, dem Benutzer, dessen Session entwendet werden soll, eine dem Angreifer bekannte Session-ID zukommen zu lassen, die dieser dann verwendet. Dies kann auf unterschiedliche Art und Weise geschehen.

Die einfachste Methode ist es, dass sich der Angreifer eine beliebige Session-ID aussucht und das Opfer dazu bringt, auf die gewünschte Seite zuzugreifen, jedoch über einen vom Angreifer erstellten Link. Der leichteste Weg ist die Mitgabe der Session-ID über einen *GET*-Parameter. Die Web-Applikation erstellt nun eine neue Session mit der angegebenen Session-ID und sobald sich der Benutzer angemeldet hat, kann die Session durch die Kenntnis der Session-ID bernommen werden.

Viele Session-Management-Implementationen verhalten sich glücklicherweise nicht so. Sie akzeptieren nur die von ihnen erstellten Sessions-IDs.

Dies ist jedoch auch nicht viel schwerer zu umgehen, da der Angreifer lediglich eine neue Session erstellen muss, bevor er den gefälschten Link erzeugt. Somit muss sich der Angreifer innerhalb des Session-Timeouts anmelden, damit keine neue Session angelegt wird.

Eine sehr effektive Methode, es dem Angreifer möglichst schwer zu machen, ist es, die Session-ID nur über Cookies zu transportieren, da durch die Same Origin-Policy das Injizieren eines Session-ID-Cookies in den Browser des Opfers deutlich erschwert wird.

Im Gegensatz zum Session Hijacking (siehe voriger Abschnitt) stellt die Verschlüsselung des Datenverkehrs keinen Schutzmechanismus dar.

5 Directory Traversal

Beim sogenannten *Directory Traversal* nutzt ein Angreifer Schwachstellen in HTTP-Servern oder darauf laufenden Applikationen, um auf Dateien zuzugreifen, die eigentlich nicht für ihn bzw. den Webserver zugreifbar sein sollten.

Webserver liefern ihre Dateien aus einem sogenannten *DocumentRoot* heraus aus. Oberhalb dieses Verzeichnisses sollte der Webserver den Zugriff auf Dateien und Ordner verweigern, da unter Umständen wichtige Konfigurationsdateien oder andere Daten ausgelesen werden könnten.

Der Webserver sollte seinen Dateisystemzugriffen immer seine DocumentRoot voranstellen: ein Zugriff auf `http://www.example.com/index.php`, der für den Server selbst einen Zugriff auf `/index.php` bedeutet (wobei `/` hier das DocumentRoot darstellt), muss im Dateisystem ausgeführt werden als Zugriff auf `/var/www/index.php`. Dies ändert jedoch nichts an der Verwundbarkeit.

Fügt ein Angreifer Strings wie `../` oder Umschreibungen dieser Zeichenkette in seine Serveranfrage ein, so kann er aus dieser Beschränkung ausbrechen.

6 Remote File Inclusion (RFI)

Diese Schwachstelle, für die (fehlkonfiguriertes) PHP besonders anfällig ist, ermöglicht es dem Angreifer, beliebigen Code in eine Applikation einzubinden. Meist bezeichnet *Remote File Inclusion* das Einbinden von Dateien des Angreifers, die auf dessen Webserver oder dem eines kostenlosen Webhosters liegen, welche dann vom angegriffenen Server ausgewertet werden, als wären sie Teil der dort laufenden Applikation. Da die Auswertung serverseitig passiert, werden die Attacken mit allen Rechten der ausgenutzten Anwendung ausgeführt und haben Zugriff auf die entsprechenden Daten wie Dateien und Umgebungsvariablen des Servers, Benutzersessions, etc. Passiert das Auswerten clientseitig, so fällt der Angriff meist in die bereits vorgestellte Kategorie Cross Site Scripting (XSS).

7 SQL Injection

SQL Injection ist eine Schwachstelle, die, unabhängig von der Programmiersprache, auf interaktiven Webseiten mit Datenbankanbindung auftreten kann. Werden die Benutzereingaben nicht ausreichend geprüft und in der Datenbankanfrage verwendet, so kann ein Angreifer SQL-Code einschleusen. Auf diese Weise ist nicht nur das Ausspähen der Datenbankinhalte möglich - da in SQL-Datenbanken lange Zeit brauchbares Benutzermanagement fehlte und es sich auch nach dessen Einführung nur langsam durchsetzt, läuft selbst Software, die eigentlich nur Lesezugriff auf die Datenbank benötigen würde, mit Schreibrechten und jeder Angreifer ist prinzipiell in der Lage, die zur Anwendung gehörigen Datenbankinhalte komplett zu löschen, überschreiben oder verändern. Wenn der Angreifer zum Beispiel die Datenbankstruktur einer Software herausgefunden hat, die die Datenbank zur Benutzerverwaltung nutzt, so kann er sogar bei schlecht konfigurierten Webservern in der Lage sein, sich Administratorrechte zu beschaffen.

Literatur

- [HGM04] Hoglund, Greg und Gary McGraw, *Exploiting software (How to break code)*, Addison-Wesley Professional, 2004.
- [Als06] Alshantsky, Ilia (2006), *Die Gefahren von XSS und CSRF*, <http://phpsolmag.org/de/phpsolmag/download.html> (aufgerufen 12. Dezember 2006)
- [Hus04] Huseby, Sverre H., *Innocent Code: A Security Wake-up Call for Web Programmers*, Wiley & Sons, 2004.
- [Kap] Kapoor, Shray (no date), *Session Hijacking - Exploiting TCP, UDP and HTTP Sessions*, http://www.infosecwriters.com/-text_resources/-pdf/-SKapoor_SessionHijacking.pdf (aufgerufen 12. Dezember 2006)
- [Jak06] Edge, Jake, (2006), *Remote file inclusion vulnerabilities*, <http://lwn.net/Articles/203904/> (aufgerufen 12. Dezember 2006)