

**AJAX IN
ACTION**

Ajax, aber sicher!

Carsten Eilers

Vorstellung

- Berater für IT-Sicherheit
- Autor
 - About Security
 - Standpunkt Sicherheit
- Schwachstellen-Datenbank
 - „Security Aktuell“ auf entwickler.de

Agenda

- „Myth-Busting“
- Etwas Geschichte
- Ein Klassiker: XSS
- Auch ein Klassiker: CSRF
- Das ganze mit Ajax
- Gegenmaßnahmen
- Fazit
- Fragen?

„Myth-Busting“

- „Ajax vergrößert die Angriffsfläche“
- **JEIN:**
 - auf dem Server: **NEIN**
 - Ajax betrifft den Client, für den Server ändert sich (fast) nichts
 - auf dem Client: **JA**
 - Ajax-Toolkits, neue Formate wie JSON
 - XMLHttpRequest

Geschichte (1)

- Die ersten 10 Jahre
 - 1989 - WWW, textbasierte Browser
 - 1993 - Mosaic, Anzeige von Grafiken
 - 1995 - Java, Navigator 2.0 mit JavaScript
 - 1996 - ActiveX (ehemals OLE 2.0)
 - 1997 - Flash 1
 - 1999 - Dotcom-Blase platzt (März 2000)

Geschichte (2)

- 1999:
 - Browser sind (dumme) Clients
 - Sicherheit = Schutz der Server vor böartigen Clients

Geschichte (3)

- Die nächsten 10 Jahre
 - 1999 - Dotcom-Blase platzt (März 2000)
 - 2005 - Jesse James Garret „findet“ Ajax
 - 2007 - iPhone: Anwendungen als Webanwendungen
 - 2009 - Browser-Rootkits?

Geschichte (4)

- 2007:
 - Browser sind mächtige Plattformen
 - Sicherheit = zusätzlich Schutz der Anwendungen auf dem Client voreinander

Agenda

- „Myth-Busting“
- Etwas Geschichte
- **Ein Klassiker: XSS**
 - **Arten**
 - **DOM-basiertes XSS**
 - **Folgen von XSS**
- Auch ein Klassiker: CSRF
- Das ganze mit Ajax
- Gegenmaßnahmen

Ein Klassiker: XSS (1)

- CVE-Datenbank:
 - 2001: 2,2% (Platz 11)
 - 2002: 8,7% (Platz 2)
 - 2003: 7,5% (Platz 2)
 - 2004: 10,9% (Platz 2)
 - 2005: 16,0% (Platz 1)
 - 2006: 18,5% (Platz 1)
 - Insgesamt: 13,8% (Platz 1)

Ein Klassiker: XSS (2)

„Same-Origin-Policy“:

Von einer vertrauenswürdigen Webseite mitgelieferter JavaScript-Code ist vertrauenswürdig

Durch XSS verletzt

Ein Klassiker: XSS (3)

3 Arten von XSS:

- DOM-basiertes XSS
Einschleusen über präparierte URL
- Reflektiertes XSS
Einschleusen über präparierte URL oder Formular
- Persistentes XSS
Einschleusen über z.B. Gästebuch

DOM-basiertes XSS (1)

- DOM-basiertes XSS

Hallo

```
<script>
```

```
  var pos=document.URL.indexOf("name")+5;
```

```
  document.write(
```

```
    document.URL.substring(
```

```
      pos,document.URL.length));
```

```
</script>
```

Willkommen auf dieser Seite ...

DOM-basiertes XSS (2)

- Aufruf:
`http://www.server.example/index.html?
name=Alice`
- XSS:
`http://www.server.example/index.html?
name=<script>alert('XSS')</script>`

DOM-basiertes XSS (3)

- Wieso?
 - Browser sendet Request an Server
 - empfängt statische HTML-Seite
 - parst Daten ins DOM
 - kopiert Teil der URL in HTML-Seite
 - parst die fertiggestellte Seite
- Nicht bei Mozilla und Co:
< und > werden zu %3C bzw. %3E

DOM-basiertes XSS (4)

Schadcode tarnen:

- als Fragmentbezeichner:
`/index.html#name=<script>alert('XSS')</script>`
- durch falschen Parameter:
`/index.html?noname=<script>alert('XSS')</script>`
- durch zusätzlichen Parameter:
`/index.html?nix=name=<script>alert('XSS')</script>
&name=Alice`

Folgen von XSS (1)

- Wer JavaScript ausführen kann, kontrolliert den Browser:
 - Cookies, Tastatureingaben, Mausevents ausspähen
 - Passwörter aus Password-Safe / Schlüsselring auslesen
 - Browser-History ausspähen
 - Klassisch: Falsche Informationen einfügen

Folgen von XSS (2)

- Wer den Browser kontrolliert, ist hinter der Firewall:
 - Portscan im lokalen Netz
 - Ausnutzen von Default-Passwörtern
 - CSRF-Angriffe gegen lokale Dienste/Server

Der Angriff kommt von innen!

Folgen von XSS (3)

- Wer den Browser kontrolliert, ist nur einen Schritt von der Kontrolle über das System entfernt
 - Symantec Internet Security Threat Report, 1. Jahreshälfte 2007:
 - 39 Lücken im Internet Explorer, 34 in Mozilla, 25 in Safari, 7 in Opera
 - 237 Lücken in Plug-ins (davor 74)
 - 89% in ActiveX-Komponenten (davor 58%)
 - > 50% geschlossener Lücken in Browsern

Agenda

- „Myth-Busting“
- Etwas Geschichte
- Ein Klassiker: XSS
- **Auch ein Klassiker: CSRF**
- Das ganze mit Ajax
- Gegenmaßnahmen
- Fazit
- Fragen?

Auch ein Klassiker: CSRF (1)

- Authentifizierung nicht für jede Aktion, sondern Speicherung des Benutzerstatus (z.B. Cookie, HTTP-Basic-Authentication)
- Status wird vom Browser automatisch mit jedem Request mitgeschickt
- Auch, wenn nicht der Benutzer, sondern ein Skript den Request auslöst

Auch ein Klassiker: CSRF (2)

- Ein Beispiel:
 - Authentifizierung über Cookies,
 - Benutzer anlegen über die URL
`http://www.server.example/adduser.php?name=X&password=Y`
- CSRF-Angriff:
 - Seite mit img-Tag
``

Auch ein Klassiker: CSRF (3)

- Eingelogger Administrator öffnet die Seite
- Browser sendet GET-Request zum Laden des Bilds
- Browser sendet Authentifizierungs-Cookie mit

=> Der Benutzer wird angelegt

Agenda

- Ein Klassiker: XSS
- Auch ein Klassiker: CSRF
- **Das ganze mit Ajax**
 - **Vorbemerkungen**
 - **Einmal mischen, bitte: Mashups**
 - **JavaScript-Hijacking**
 - **Alles zusammen: JavaScript-Malware**
- Gegenmaßnahmen

Das Ganze mit Ajax (1)

- Sicherheitskonzept Same-Origin-Policy: XMLHttpRequests nur an Server der Seite
- Umgehen z.B. durch Proxy
 - Proxy für Mashups
 - Google Translate

Das Ganze mit Ajax (2)

Weitere Risiken:

- XMLHttpRequest im Hintergrund
- JSON wird in eval() ausgewertet
- JSON kann entführt werden

Mashups (1)

„Einmal mischen, bitte“

Grundsatz #1:

Nie Daten vom Client vertrauen!

Und was ist mit Daten von anderen Servern?

Mashups (2)

- Mashups tun, was die Same-Origin-Policy verhindern soll:
Zugriffe des Codes von Seite A auf Seite B
- Proxy untergraben Same-Origin-Policy:
`http://www.server-a.example/proxy?url=
http://www.server-b.example/die-daten`

Mashups (3)

2 Probleme:

- Angreifer können Herkunft verschleiern oder Same-Origin-Policy unterlaufen
- Code von Server B läuft in Seite von Server A
 - in deren Kontext
 - mit eigenen Schwachstellen

JavaScript-Hijacking (1)

- Beispiel:
`http://www.server.example/daten.json`
- Angreifer lockt Opfer auf Seite mit überschriebenen Objekt-Konstruktor
- Lädt JSON-Daten über script-Tag, Browser sendet Cookie mit
- Konstruktor sendet Daten an Angreifer

JavaScript-Hijacking (2)

Funktioniert nur, wenn beim Erkennen eines Literals der Array- oder Objekt-Konstruktor aufgerufen wird

Zur Zeit nur Mozilla und seine Ableger

JavaScript-Malware (1)

- Web-2.0-Würmer sind Realität
- Lebensraum: WWW / Webanwendungen
- Infizierung: XSS, CSRF
- Verbreitung: Ajax, Formulare, Links, RSS
- Über Mashups in andere Anwendungen
- Plattformunabhängig
- Können weitere Payload mitbringen

JavaScript-Malware (2)

- Beispiel: MySpace-Wurm Samy
 - Ausgehend von Samys Profil in 20 Stunden in über 1 Million Profile verbreitet
 - MySpace filterte die Eingaben, aber Samy fand eine Möglichkeit zum Umgehen der Filter
 - Ablauf:
 - Benutzer besucht befallenes Profil
 - XMLHttpRequest macht Samy zum Freund+Hero
 - Wurmcode wird in Profil des Benutzers kopiert

JavaScript-Malware (3)

- Beispiel: Jikto
 - von SPI Dynamics für Demo-Zwecke entwickelter Schwachstellenscanner für JavaScript
 - Gelangte in „freie Wildbahn“
 - ca. 875 Zeilen kommentierter JavaScript-Code
 - kann beliebige Webseiten durchsuchen
 - Umgeht Same-Origin-Policy über Google Translate

JavaScript-Malware (4)

- Beispiel: Browser Exploitation Framework
 - läuft auf Server
 - Clients verbinden sich mit Server, danach können verschiedene Angriffe ausgeführt werden
 - Portscan
 - Clipboard lesen
 - Keylogger
 - Shell öffnen

Agenda

- Ein Klassiker: XSS
- Auch ein Klassiker: CSRF
- Das ganze mit Ajax
- **Gegenmaßnahmen**
 - Vorbemerkungen
 - Verhindern von XSS
 - Verhindern von CSRF
 - Verhindern von JavaScript-Hijacking
- Fazit

Gegenmaßnahmen

Vertraue nie dem Client!

Prüfungen **auf** dem Client **für** den Client
(Bequemlichkeit des Benutzers)

Prüfungen **auf** dem Server **für** den Server
(Sicherheit des Servers)

XSS verhindern (1)

- Filter sollen JavaScript ausfiltern, aber HTML (meist) unbeschädigt lassen
- Filter können umgangen werden:
„XSS Cheat Sheet“
- Neue Schwachstellen oder Exploits umgehen vorhandene Filter
- Wettlauf von Hase und Igel

XSS verhindern (2)

- DOM-basiertes XSS:
 - Vorsicht bei Manipulationen der HTML-Seite oder des DOM, wenn die Daten manipuliert sein können
 - Prüfung auf dem Client - der Server sieht den Schadcode evtl. gar nicht
 - Wenn Änderungen nicht vermeidbar, z.B. `.innerText` statt `.innerHTML` nehmen
 - `eval()` ist evil

XSS verhindern (3)

- Daten auf dem Server passend vorbereiten
- Client-Logik darf keinen Einfluss auf Sicherheit oder Business-Logik haben
- Möglichst fertige Frameworks statt Eigenentwicklungen verwenden
Für Eigenentwicklungen interessieren sich nur Sie und alle Angreifer

CSRF verhindern (1)

- Gefahr besteht, wenn statische GET- oder POST-Request verwendet werden
- Test auf Anfälligkeit:
Benutzung 2x über Proxy aufzeichnen,
Aufzeichnungen vergleichen: Keine
Änderungen der Request? Gefahr!

CSRF verhindern (2)

- Gegenmaßnahme:
Nicht vorhersagbares Token in jedes Formular, nur Request mit gültigen Token ausführen
- Nutzlos bei XSS-Schwachstelle
- Zusätzlich:
Rückfrage oder erneute Passwortabfrage vor gefährlichen Aktionen, ggf. mit CAPTCHA

JSON-Hijacking verhindern

- Böartige Requests nicht ausführen:
Analog zum Vorgehen bei CSRF
- Direkte Ausführung verhindern:
 - while(1) am Anfang schickt Angreifer in Endlosschleife
 - Kommentarzeichen lassen Angreifer nichts zum Auswerten

Allgemein (1)

- Sichere Entwicklung:
 - Fallstricke kennen und vermeiden:
 - Prüfung fremder Daten, möglichst mit Positivlisten (Samy läßt grüßen!)
 - Server weiß nicht, wer den Request gesendet hat
 - Passenden Character-Set definieren
z.B. ASCII-Seite mit UTF-7-XSS-Code
 - Sichere Frameworks etc. verwenden
 - Schwachstellen-Scanner nutzen

Allgemein (2)

- Sichere Konfiguration aller Komponenten
- Web Application Firewall als zusätzliche Hürde
- Tests wiederholen
 - Systemänderungen
 - neue Schwachstellen oder Exploits

Fazit

- Web 1.0: Schutz der Server
- Web 2.0: Schutz auch für den Client
- Mächtige Clients ergeben mächtige Angreifer
- Finger weg von ungeprüften Daten



Fragen?



Vielen Dank...

... für Ihre Aufmerksamkeit!

Material und Links auf
www.ceilers-it.de/konferenzen/