



## Web 2.0 (In) Security

PHPUG Würzburg | 29.06.2006 | Björn Schotte



Alte Freunde – SQL Injections, Code Executions & Co.

Cross Site Scripting

Cross Site Scripting in der Praxis

JavaScript Security: Broken by Design

Warum XSS im Web2.0 mehr schmerzt

- XMLHttpRequest (Anfragen im Hintergrund)

- „move more power to the client“ (Client-Logik)

- Noch relativ neue „Technik“ – viel Unvorsichtigkeit

Ajax, Dojo, Prototype und Co

Sicher 2.0

Tools & URLs

# Alte Freunde aus dem Web 1.0

Sichere Webserver gibt es durch

Offline

Aus

Risiken bei Webapplikationen

Remote Code Execution

Privilege Escalation

Information Disclosure

Content Manipulation

Denial of Service

Klassische Angriffsvektoren

Schlechte Programmierung

SQL Injections

Code Executions/Inclusions

Cross Site Scripting (XSS)



# Cross Site Scripting

Siehe Wikipedia (Einschleusen von Informationen aus einem nicht-vertrauenswürdigen Kontext in einen vertrauenswürdigen Kontext. Daraus Angriffe möglich)  
Nutzer können also JavaScript in eine Applikation einschleusen

Sehr populärer Angriffsvektor – siehe Meldungen auf Bugtraq, heise Security & Co.

Missbrauch:

- Authorisierungsdiebstahl („Huch, ich bin Admin“)

- Content Manipulation

- Datenmanipulation

Jeder ist betroffen: Google, Yahoo Mail, del.icio.us, Flickr & co.

## MySpace Wurm

Online Community mit >30.000.000 Nutzern

Ins eigene Profil konnte JS eingeschmuggelt werden

Mit diesem JS wurde man als Kontakt eingetragen

Weiterverbreitung über fremde Profile

In 20 Std. hatte der Wurm mehr als 1 Mio. Accounts getroffen. Heute, 1,5 Jahre später, sind immer noch >75.000 Nutzer betroffen

## Del.icio.us

Auf einem RSS Aggregator ein JS eingeschmuggelt, das über einen XSS auf del.icio.us den Login-Cookie kopiert hat

# JavaScript Security: broken by design

Fast alles kann überschrieben werden

(existierende) Methoden

Funktionen

Variablen

Fast allem wird vertraut:

```
<script src=„http://192.168.0.1/config.js“ > </script>
```

```
<form action=„http://fritz.box/cgi-bin/webcm“ >
```

WLAN Login Hack auf PHP Conference 2001 (social engineering)

Cookies/Auth nach Browserinstanz

Wenn ein Nutzer auf Domain X eingeloggt ist, kann Domain Y Formulare unter seinem Login abschicken

## Warum XSS im Web2.0 mehr schmerzt

Mehr Funktionalitäten im JS („move more power to the client“)

Persistente Seiten bei AJAX: ein XSS-Angriff bleibt für immer (Laufzeit der Applikation) erhalten

JS ist ein Hackerparadies: nach XSS-Manipulation ist in JS nichts mehr vertrauenswürdig

Always logged on

Auf kollaborativen Plattformen, GMail & Co. Loggt man sich nicht täglich neu ein

Erleichtert Authorisierungsdiebstahl und Datenmanipulation

## Kein X in AJAX

Standard Datenkodierung ist nicht XML, sondern JSON  
JSON wird in JS evaluiert – damit wird enthaltener  
JavaScript Code ausgeführt („eval is evil“)

Beispiel: Prototype hat keinerlei Sicherheitsüberprüfung

JS Erweiterungen sind Filter-Evasions:

Dojo Widgets gehen an jedem JS-Filter vorbei

xmlHttpRequest: Cookie Diebstahl ohne  
Browsernotifikation

Vollständige Validierung der Daten

- Datenbank nur mit offiziellem Escaping

- Oder besser: Binding

- HTML Input kann man nicht sichern (CMSe, Blogs etc.)

- Alles andere auf Zeichen-Basis validieren

- XSS-Filter + Tainted Mode

Security is a process, not a product

- Es gibt keine absolute Sicherheit

- Es gibt viele unbekannte Bugs

  - Beispiel: Google UTF-7 Bug

  - on\0mouseover

CSRF Schutz!

- Formulare und Ajax-Übergaben sollten per Token abgesichert werden

## Nützliche Tools & URLs

XSS – ein XSS-Scanner ( <http://www.sven.de/xsss/> )

XSS Cheat Sheet ( <http://ha.ckers.org/xss.html> )

OWASP Web Security Top 10 ( <http://www.owasp.org/documentation/topten.html> )

Eigenwerbung 😊 (leckere spanische Wurst auf Nachfrage)



**Vielen Dank für Ihre Aufmerksamkeit**

Björn Schotte

Mayflower GmbH

Pleichertorstr. 2

97070 Würzburg

+49 (931) 35 9 65 - 0

[schotte@mayflower.de](mailto:schotte@mayflower.de)

