Cross-Site Request Forgery

Andy Podgurski
EECS Dept.
Case Western Reserve University

Sources

- *Cross-Site Request Forgeries: Exploitation and Prevention* by W. Zeller and E. W. Felten, 2008
- *Browser Protection Against Cross-site Request Forgery* by W. Maes et al, Workshop on Secure Execution of Untrusted Code, 2009

Cross-Site Request Forgery (CSRF) Attacks

- These occur when a malicious web site causes a user's browser to perform an unwanted action on a trusted site
- XSS defenses *do not* protect against CSRFs

Example: Web-Based Email Site

- Site uses *implicit authentication* of users
- One page contains form allowing user to create and send an email
  - **`http://example.com/compose`**

Example cont. (2)

```
<form
action="http://example.com/send_email.htm"
method="GET">
Recipient's Email address:  <input
type="text" name="to">
Subject:  <input type="text" name="subject">
Message:  <textarea name="msg"></textarea>
<input type="submit" value="Send Email">
</form>
```

Example cont. (3)

- When user clicks “Send Email”, the data he entered is sent to http://example.com/send_email.htm as a GET request, e.g.,

```
http://example.com/send_email.htm?to=bob%
40example.com&subject=hello&msg=What%27s+the+
status+of+that+proposal%3F
```

Example cont. (4)

- The page **`send_mail.htm`** sends an email from the user to the recipient
- It doesn't verify that the data originated from the form on **`compose.htm`**
  - If the user typed the URL above into his browser, **`example.com`** would still send an email
- This makes a CSRF attack possible

Example cont. (5)

- A CSRF attack occurs if an attacker gets the user to send a request to **`send_email.htm`** that causes **`example.com`** to send an email with data the attacker chose
- Assume the user visits a site controlled by the attacker
- The attacker needs to forge a *cross-site request* from his site to **`example.com`**
- HTML provides many ways to make such requests

Example cont. (6)

- The **<img>** tag will cause the brower to load whatever URI is set as the **src** attribute
- The attacker can create a page with:

```
<img src="http://example.com/send_email.htm?
to=mallory%40example.com&subject=Hi&msg=My+
email+address+has+been+stolen">
```

Example cont. (7)

- When the user visits the page, a request will be sent to **`send_email.htm`**
- It will then send an email to Mallory from the user
- CSRF attacks are successful when an attacker can cause a user's browser to perform an unwanted action on another site
- They are typically *as powerful as the user*

Same-Origin Policy Doesn't Help

- The *same-origin policy* was designed to prevent an attacker from accessing data on a *3rd-party site*
- It does not prevent requests from being sent
- Hence, it doesn't protect against CSRF attacks

Authentication & CSRF

- CSRF attacks often *exploit authentication mechanisms* of sites
- Web authentication ensures that a request came from a certain user's browser
- It does *not ensure* that the user actually made or authorized the request

Example Attack

- Alice logs on to a target site T
- T gives Alice's browser a cookie with a session ID *sid* and records that Alice is logged into session *sid*
- Alice visits a malicious site M
- Content supplied by M contains JavaScript or an image tag
- It causes Alice's browser to send an HTTP request to T
- Alice's browser "helpfully" appends *sid*
- T performs the requested operation on Alice's account

Authentication cont.

- Most web authentication mechanisms suffer from the same problem
  - HTTP BasicAuth
  - Client-side SSL certificates
  - IP-addressed-based authenticatioin
- Whenever authentication happens *implicitly* there is a CSRF risk

CSRF Attack Vectors

- For attack to work, the user must
  - Be *logged-in* to the *target site*
  - *Visit* the *attacker's site* or a site that attacker partially controls
- If the server is vulnerable and accepts GET requests, CSRF attacks are possible *without use of JavaScript*
- If the server accepts only POST requests, *JavaScript is required* to send a POST from the attacker's site to the target site

CSRF vs. XSS

- An XSS attack occurs when an attacker injects malicious JavaScript into a site for the purpose of targeting other users of the site
- CSRF does not require JavaScript
- With CSRF, malicious code is located on *3rd party sites*
- Filtering user input will not prevent malicious code from running on 3rd party sites
- *Protection from XSS does not protect against CSRF*
- *If a site is vulnerable to XSS, it is also vulnerable to CSRF*

Example: NYTimes.com Vulnerability

- Can be used to determine member's email address
- Exploits the "Email This" feature, which sends a email like this:

This page was sent to you by: **[USER'S EMAIL ADDRESS]**

Message from sender:
Thought you'd be interested in this.

NATIONAL DESK
Researchers Find Way to Steal Encrypted Data
By JOHN MARKOFF
A computer security research group has developed a way to steal encrypted information from computer hard disks.

NYTimes.com cont. (2)

- Attacker causes logged-in user's browser to send a request to the "Email This" page
- This page does not protect against CSRF attacks
- It will send an email to an address chosen by the attacker
- If the attacker changes the *recipient* address to his own email address, he'll receive an email from NYTimes.com containing the user's email addres

NYTimes.com cont. (3)

- Each Times article contains a link to the “Email This” page, which contains form, with hidden variables, e.g.,

```
<form
action="http://www.nytimes.com/mem/emailthis.html"
method="POST"
enctype="application/x-www-form-urlencoded">
<input type="checkbox" id="copytoself"
name="copytoself" value="Y">
<input id="recipients" name="recipients"
type="text" maxlength="1320" value="">
<input type="hidden" name="state" value="1">
<textarea id="message" name="personalnote"
maxlength="512"></textarea>
<input type="hidden" name="type" value="1">
<input type="hidden" name="url"
value="[...]">
```

NYTimes.com cont. (4)

- The attacker can convert the form into a GET request that can be used in an **<img>** tag
- The attacker make the URL the **src** attribute of an **<img>** tag
- If a logged-in user visits any page containing this tag, the browser will load the “Email This” page with the attacker’s parameters
- This will cause the site to send an email to the attacker containing the user’s email address

Server-Side Protection

- Allow GET requests to only *retrieve* data, not modify any data on the server
- Require all POSTs to include a *pseudorandom cookie*
  - In a valid request, the form value and the cookie should be the same
  - When an attacker submits a form on behalf of the user, he can only *modify* values of the form
  - He can't read data sent from the server or modify cookie values
  - The attacker will be unable to submit a valid form unless he can *guess* the cookie value
- Use a pseudorandom value that is independent of the user's account

Client Side Protection

- Zeller & Felten developed a Firefox extension that intercepts every web request and decides if it should be allowed
  - Any non-POST is allowed
  - If the requesting site and target fall under same-origin policy, the request is allowed
  - If the requesting site can make a request to the target using *Adobe's cross-domain policy*, the request is allowed