21

WEB SECURITY: CROSS-SITE REQUEST FORGERY (CSRF)

HTML forms

- allow users to apply input to a web application

```
<h1>Sign in</h1>
<form action="login.php" method="GET">
  <div><input type="text" name="username" placeholder="username"/></div>
  <br>
  <div><input type="password" name="password" placeholder="password"/></div>
  <br>
  <div><input type="submit" value="Sign me in" /></div>
</form>
```

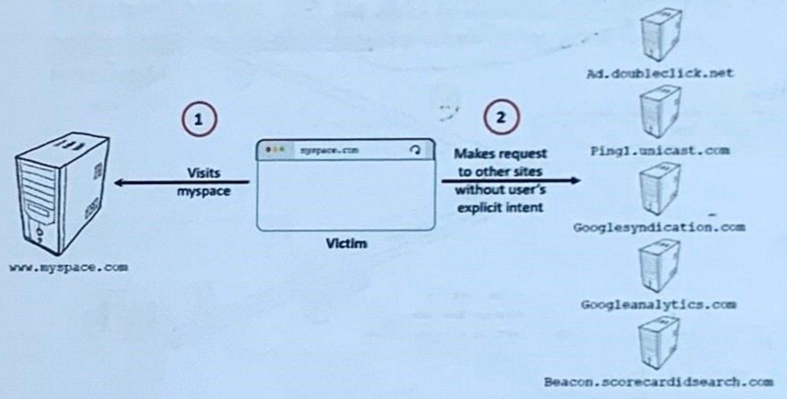
→ Upon clicking submit the browser issues the request; (a GET request)

```
GET /login.php?username=Myrto&password=123456 HTTP/1.1
Host:...
```

The browser will include all relevant cookies.

↖ whenever browser issues a HTTP request to a URL

How the web works



CSRF attacks

OWASP

CSRF forces a user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

Target: user who has an account on vulnerable web application

Main steps of attack:

1. build an exploit URL
2. trick the victim into making a request to the vulnerable server as if intentional

Attacker tools:

1. ability to get the user to "click exploit link"
2. ability to have the victim visit attacker's server while logged-in to vulnerable server

Keys ingredient: requests to vulnerable server have predictable structure

CSRF: a simple example

1. Alice logs in to bank.com and gets a session cookie
2. bank.com form for requesting transfers:

```
<form action="http://bank.com/transfer.php"
method="GET">                          account
  <input type="text" name="acct" />
  <input type="text" name="amount" />
  <input type="submit"/>           amount
</form>
```

3. Alice visits evil.com which included

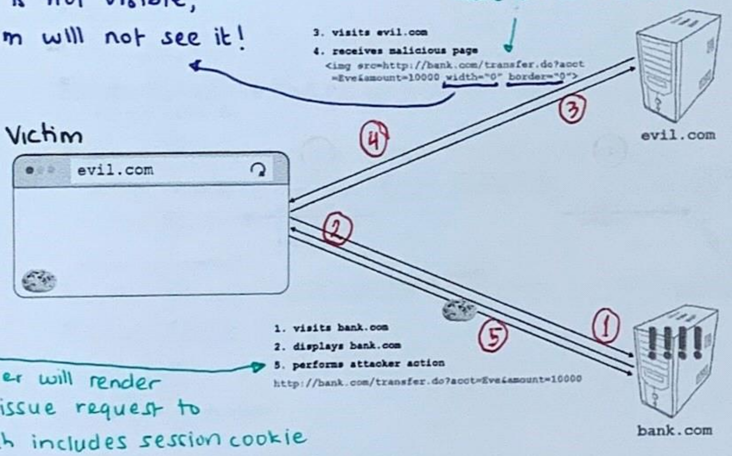
```
<form name="attack"
action="http://bank.com/transfer.php" method="GET">
  <input type="text" name="acct" value="Eve"/>     - PRE-FILLED
  <input type="text" name="amount" value="100000"/>  FORM
</form>
<script> document.attack.submit(); </script>  - a Javascript thr will automatically click on 'submit' button
```

4. Alice's browser sends the session cookie along with the HTTP GET request !! which is why the transfer will go through

CSRF flow

This picture is not visible,
hence victim will not see it!

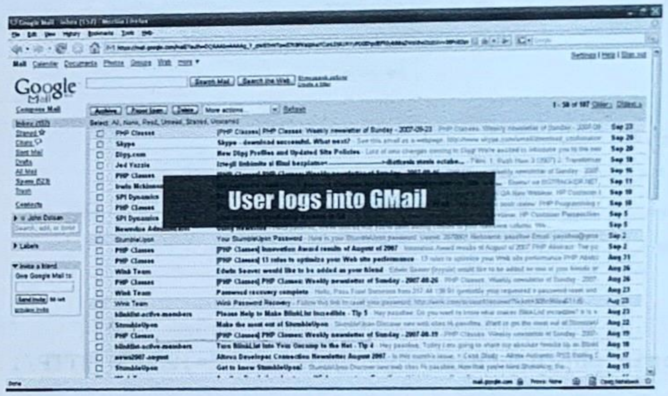
somewhere in the page,
there is this line here


Victim
evil.com
3. visits evil.com
4. receives malicious page
<img src=http://bank.com/transfer.do?acct =Eve&amount=10000 width="0" border="0">
evil.com
1. visits bank.com
2. displays bank.com
5. performs attacker action
http://bank.com/transfer.do?acct=Eve&amount=10000
bank.com
1
2
3
4
5

Victim's browser will render
the page and issue request to
bank.com which includes session cookie

Cookies are insufficient when side effects are present at server-side

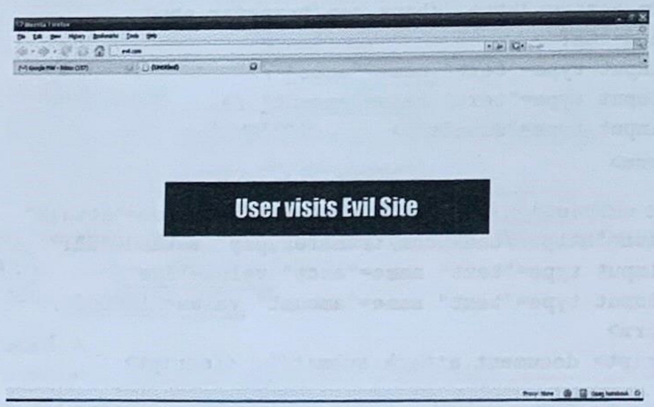
Gmail filter (2007) - step 1



www.davidairey.com/google-gmail-security-hijack/

Gmail filter (2007) - step 2

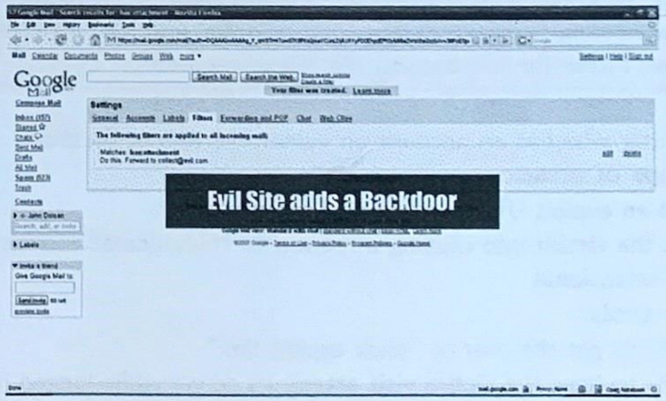
evil.com contains Gmail CSRF attack code



www.davidairey.com/google-gmail-security-hijack/

Gmail filter (2007) - step 3

User submits request to Gmail, creating a filter to forward all mail to hacker



Your filter was created. Learn more

Settings

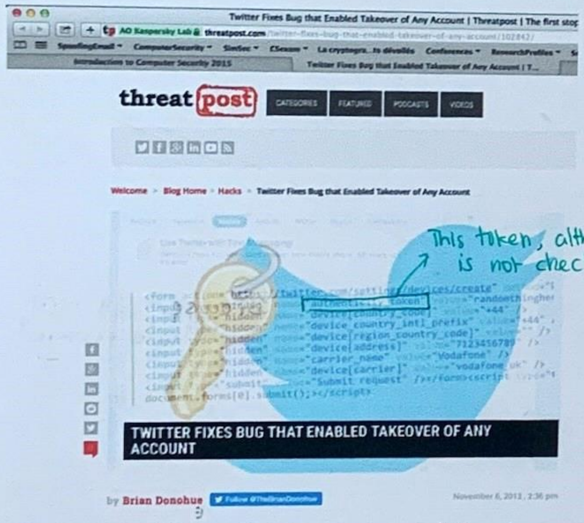
The following filters are applied to all incoming mail

Matches: has:attachment
Do this: Forward to collect@evil.com

www.davidairey.com/google-gmail-security-hijack/

CSRF DEFENSES

Twitter SMS account hijacking (Nov. 2013)



Welcome > Blog Home > Hacks > Twitter Fixes Bug that Enabled Takeover of Any Account

```
<form [illegible]="https://twitter.com/settings/devices/create" [illegible]
<input [illegible] "authenticity_token" [illegible]="randomthingher
<input [illegible] hidden [illegible] "device[country_code]" [illegible]="+44" />
<input [illegible]hidden" [illegible]"device_country_intl_prefix" [illegible]"+44" ,
<input [illegible]hidden" [illegible]"device[region_country_code]" [illegible]"" />
<input [illegible]hidden" [illegible]"device[address]" [illegible]"7123456789" />
<input [illegible]hidden" [illegible]"carrier_name" [illegible]"Vodafone" />
<input [illegible]hidden" [illegible]"device[carrier]" [illegible]"vodafone_uk" />
<input [illegible]"submit" [illegible]"Submit request" /></form><script [illegible]
document.forms[0].submit();></script>
```

TWITTER FIXES BUG THAT ENABLED TAKEOVER OF ANY ACCOUNT

by Brian Donohue

November 6, 2013, 2:36 pm

This token, although there, is not checked!

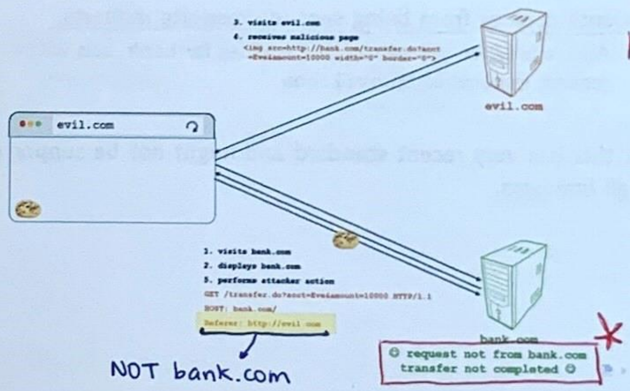
https://henryhoggard.co.uk/blog/
The-bug-that-let-me-tweet-from-any-account

1. Check the referer

- The client's HTTP request includes the referer header specifying the context from which this request was issued
- The server ensures that the HTTP request has come from the original site means that attacks from other sites will not function

does not require per-user state; useful when memory is scarce

Raises privacy issues!!



```
3. visits evil.com
4. receives malicious page
   <img src=http://bank.com/transfer.do?acct
   =Eve&amount=10000 width="0" border="0">
```

```
1. visits bank.com
2. displays bank.com
5. performs attacker action
GET /transfer.do?acct=Eve&amount=10000 HTTP/1.1
HOST: bank.com/
Referer: http://evil.com
```

NOT bank.com

☹ request not from bank.com
transfer not completed ☹

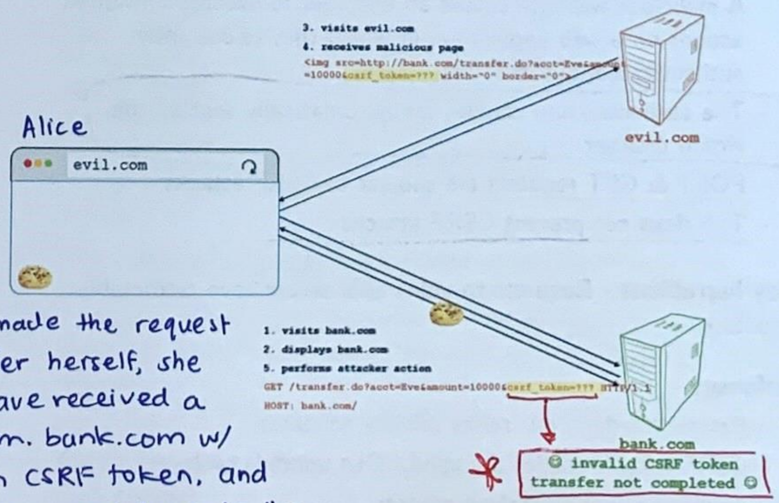
2. CSRF tokens

The idea

Make URLs unpredictable

- The server stores CSRF token along user's session token
- Includes a fresh CSRF token in every form as a hidden field
- On every request, the server checks that the supplied CSRF token is the valid one
- Must be unpredictable!
- Ruby on Rails embeds secrets in every link automatically
- To avoid any replay attack should be different in each server response

CSRF tokens



```
3. visits evil.com
4. receives malicious page
   <img src=http://bank.com/transfer.do?acct=Eve&amount
   =10000&csrf_token=??? width="0" border="0">
```

```
1. visits bank.com
2. displays bank.com
5. performs attacker action
GET /transfer.do?acct=Eve&amount=10000&csrf_token=??? HTTP/1.1
HOST: bank.com/
```

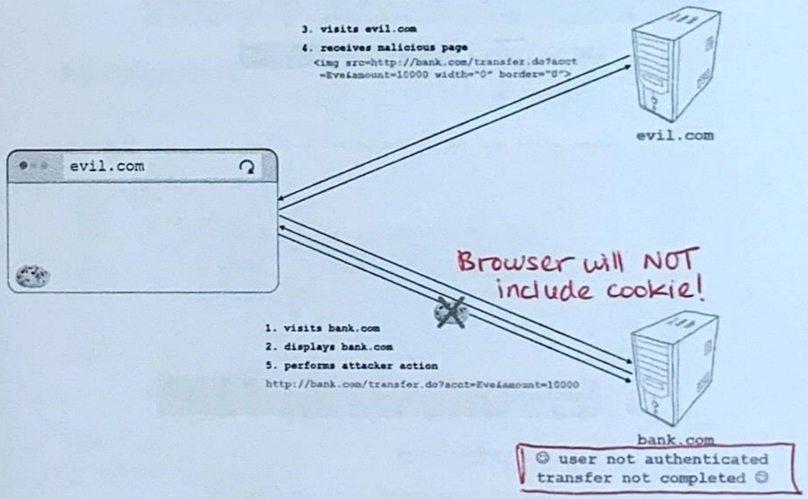
☹ invalid CSRF token
transfer not completed ☹

If Alice made the request to transfer herself, she would have received a form frm. bank.com w/ a fresh CSRF token, and this will be sent back to the bank.com where it would check it.

3. SameSite cookie attribute

- Set the SameSite flag on cookies.
- Prevents cookies from being sent in cross-site requests.
  - Alice's browser will not include cookies for `bank.com` when request issue while on `evil.com`
- But this is a very recent standard and might not be supported by all browsers.

SameSite cookie attribute



3. visits evil.com
4. receives malicious page
<img src=http://bank.com/transfer.do?acct=Eve&amount=10000 width="0" border="0">
evil.com
evil.com
Browser will NOT include cookie!
1. visits bank.com
2. displays bank.com
5. performs attacker action
http://bank.com/transfer.do?acct=Eve&amount=10000
bank.com
☹ user not authenticated transfer not completed ☹

Take away

CSRF attack - CSRFs exploit a web sites trust of a specific user.

- A malicious web site causes an end user to execute unwanted actions on a web application in which they're currently authenticated, and that they trusts.
- The authentication cookies are automatically sent by the victim browser.
- POST & GET requests are subject to CSRF attacks
- TLS does not prevent CSRF attacks

Key ingredient - Requests to vulnerable server have predictable structure

Defenses -

- Referer header - but raises privacy concerns
- CSRF token - render the valid URLs unpredictable
- SameSite authentication cookies