Cross Site Request Forgery Attacks (CRSF)

-Similar to confused deputy problem

You login, it sets a cookie, you fill out a form to transfer money then you click submit:

https://bank.com/xfer?from=... &to=...&amount=........

If you’re not logged in, and you click a similar link sent by a malicious attacker this link it won’t establish a cookie, but if you are logged in it will set a cookie.

-Browser automatically sends correct authentication cookie w/ request
- Ambient Authority problem, user clicks a link and browser tries to find the best cookie to help

How to solve the underlying problem:

1. We can add a token

https://bank.com/xfer?to=...&amount=...&token=<bigrandom> << Legitimate Transfer Link
Now attacker will be stuck since he can’t get a valid token that matches up with the UID in the banks server.

----------------

2. A separate or additional Security Feature would be to have:
   When you send the 3rd command in the diagram above it may also have:
   Referrer:<Url of page where link w/ clicked>
   To tell the bank server that your request is originating from the website.

3. Security Feature That doesn’t require the server having a Database of Tokens to UID’s

You request to your bank that you’d like to make a transfer:

Then the bank will send you a token and a form that allows you to make a transfer:

https://bank.com/xfer?to=...&amount=...&token= MAC key k(FormID,UID,cookie)

Then when you fill out the form you have to send back the same:

https://bank.com/xfer?to=...&amount=...&token= MAC key k(FormID,UID,cookie)

If you send a token back with the wrong UID or wrong Form ID (Deposit instead of a transfer for example) the server will reject the request.

The way to prevent CSRF Summary:

1. Send and check Token
2. Check Referrers
3. Perform Checking at input / output layer of your web application

----------------------------------------------------------------------------------------------------------------------------

Path Traversal Bugs

Back in the early days of the web you could do the following thing to a web server:

http://server.com/../../../../../etc/passwd

Most of the file systems were on unix and would be organized similarly to:

// var/httpd / root/ index.html
// etc/passwd

So being in index.html you can backout all the way to / then go into /etc/passwd where the usernames are stored. or /etc/shadow where passwords are stored

What the server does internally:

filename = docroot + url;
open(filename)

Many systems have a function where you can get the real path for a particular file, so what it should say is:
realFilename = realpath(docroot + url)

if (docroot is not a prefix of realname)
    error();

some incorrect solutions are:

If you see a “..” you delete them

----------------------------------------------
chroot (Jails)

/  
/  
/var

httpd
  /
usr    docRoot
  /
bin
  /
httpd

Launcher.c
Chroot("var/httpd");
Chdir("/");
Exec("/usr/bin/httpd");

- So what Chroot does is makes httpd its root directory and nothing from inside will access anything above httpd
- So if you did a terminal command “/..” it would bring u to httpd
- If you can get root you can break out of this jail

Forced Browsing(a.k.a. forgetting to check authentication)

Just because there isn’t a link to a page doesn’t mean someone can’t just type in (guessing from the pattern of previous file names) the page name manually and go to it.

My Account .
History .
.

Admin (only shown when logged in as admin)  Non Admin Links
But then there’s always: **website/admin.html**

- So you shouldn’t be able to do a forced browsing attack/access on admin.html page

Example to prevent:

```php
admin.php

Require("auth.php");
```

So just require and `auth.php` in order to access `/admin.html`

-----------------------------

A way to check/audit for this is to do a web crawl of all the pages while logged in, and then do a web crawl of all the pages while logged in as admin and see which pages overlap in the intersection of these 2 sets. So if there’s a page that you don’t want non-admins to have access to you make sure it’s not in the intersection.

---

**Mashups**

Occurs when a web server wants to combine functionality of other web servers.
Ex: HousingMaps.com = combines craigslist housing listings with google maps.

***Not a good way of organizing a mashup:
This is a better way of organizing a mashup:

- Unfortunately up to HTML 5 this is difficult to pull off
- Same Origin Policy problem
- The JavaScript on browser can’t send an AJAX request to server B because JavaScript is from Mashup
- Even if it could, then the stuff it gets back would origin from server B so mashup couldn’t access it