GridPP
UK Computing for Particle Physics

CSRF etc

Andrew McNab

University of Manchester

MANCHESTER 1824

GridPP
UK Computing for Particle Physics

Outline

- How website logins work
- XSS and CSRF attacks
- How this affects your bank etc
- How this affects “grid websites”
- GridSite solutions

How the web works

- A web browser (Firefox) connects to a web server (Apache) and says:

  GET /hellopage.html HTTP/1.1

- The server replies:

  HTTP/1.1 200 OK

  Date: Thu, 25 Oct 2007 14:30:00 GMT

  Content-Type: text/html

  Hello!

How forms work

- Say we want to tell the server something, rather than just get a page:

  POST /helloprogram.cgi HTTP/1.1

  firstname=andrew&surname=mcnab

- The server might reply:

  HTTP/1.1 200 OK

  Content-Type: text/html

  Hello andrew mcnab!

How logins work

- We now want to login, so server knows it's really me:

  POST /loginprogram.cgi HTTP/1.1

  username=andrew&password=topsecret

- The server might reply:

  HTTP/1.1 200 OK

  Content-Type: text/html

  Set-Cookie: sessionid=1234567890

  Welcome back andrew!

Cookies

- The “cookie” I got back is a name-value pair chosen by the server, which is stored in my web browser.
- My browser remembers which website it's from.
- When I look at pages from that site, my browser mentions the cookie in case it matters:

  GET /anotherpage.cgi HTTP/1.1

  Cookie: sessionid=1234567890

- The server can show me extra options as a result.

Cookie security

- Cookie session IDs are either random numbers stored by the server, or cryptographic hashes of other info that only the server could have calculated.
- So trying to steal or make use of cookies is one of the major objectives of “front door” attacks on websites.
- For privacy and security reasons, expiry times for cookies are usually set both by servers and browsers.
- Can set browsers to delete cookies when they shut down

XSS

- XSS stands for “Cross Site Scripting” but it's really a misnomer, as it's intra-site scripting in practice.
- Let's say I have a hello form that asks for your name then says hello to that name.
- But what if I give a “name” that includes lots of HTML – maybe I can get that HTML to appear on the page instead of just a name?

XSS example

POST /helloprogram.cgi HTTP/1.1

name=website user<p>Error 574!

<form method=post>

<input type=submit value=”Report error to site”>

<input type=hidden name=action value=delete>

<input type=hidden name=file value=”hackattempts.log”>

</form>

XSS example

All you see is:

Hello website user

Error 574!

Report error to site

XSS

- But so what? You wouldn't trick yourself?
- Ah, but the attacker puts a button on **their** website that submits to helloprogram.cgi and makes the fake button.
- So what you would see is:
  - You visit a website, see button marked “To continue”
  - You get transported to another website (where you're already logged in) and see an error message and a reassuring button to report the error

XSS

- But since you're logged in, the “error report” button actually deletes an import file the attacker wanted rid of
- This might seem unlikely, but it has been used against popular sites like Google Mail, to get files not delete them
- If the attacker can identify you from the victim website, they may be able to email you and tempt you to visit their (throwaway) website with the first button.
- If you accept HTML emails, they may even put the button in the email!

Preventing XSS

- Preventing these attacks is relatively straightforward: you don't let people sneak their HTML (or Javascript) into the pages of your site.
- But that's quite hard to ensure, and if you find yourself **unexpectedly** on a website you normally login to, it's safest to close down your browser/window/tab rather than clicking on any buttons you see.

CSRF

- CSRF is Cross Site Request Forgery
- Cuts out the middleman: a direct link, button etc to the victim website's script that “does something”
- Again, it relies on the victim user having a login cookie already, and the attacker knowing how the website works.
- Since some websites will “do” things with a link rather than a button, you can even embed an action link as an image in another website, or HTML email

Preventing CSRF

- From the user's side, there isn't much you can do about CSRF: you click the fake button on the attacker website, whoosh! you're on the victim website, the damage is done
- There are techniques, like double submit cookies, that developers can use to protect against CSRF
  - Will explain later
- But CSRF is further complicated due to something called XMLHttpRequest and the way Internet Explorer works...

XMLHttpRequest

- Javascript is a C-like language that can be embedded in HTML web pages.
- XMLHttpRequest() is a Javascript function that can be used to fetch files over HTML by code embedded inside another page, without having to display them.
- Interactive websites like Google Maps and Gmail rely on XMLHTTPRequest: so it's good?
- XMLHTTPRequest can do CSRF **silently**: so it's bad too!

XMLHttpRequest

- When XMLHttpRequest was invented, people sort of realised something bad might be possible.
- So browsers follow the “Same Origin Policy” as they do for cookies: only the original website that delivered the Javascript can be contacted by XMLHttpRequest.
- Unfortunately, Internet Explorer “enforces” this by putting up a weakly worded warning that you can click to ignore.
- If you say yes, then the CSRF attack can proceed.

Moral of the story

- Don't ignore warnings in Internet Explorer:
  - **Innocuous warnings in IE may be very very serious!**
- Don't set up things so users get in the habit of ignoring security warnings
  - eg ssh key changes, wrong certificates etc.

CSRF and Grids

- Many Grid project websites use X.509 user certificates
- These are equivalent to cookies that last a year
- So now the CSRF attacker doesn't even have to be lucky and pick a time when you happen to be logged in
  - You're always logged in!
- We tried to get away from passwords because of “phishing” attacks with faked-up websites: CSRF is almost the equivalent for X.509

GridSite

- GridSite consists of
  - A grid security toolkit for C/C++
    - Parses grid security objects, like GACL policies, X.509, GSI, VOMS credentials
  - An Apache module which adds support for these credentials
    - This lets people host webservices for Grids, written in C/C++/scripts/Java etc etc.

GridSite CSRF solution

- Remember the same origin policy used with cookies:
  - Only the original website pages can “see” its cookies
- So when people arrive with an X.509 user certificate, we give them a cookie and rely on that rather than the certificate itself
- When they submit a form to “do” something, they must include the value of the cookie in the form as well as the HTTP request via Javascript - “double submission”

GridSite solution (2)

- This procedure cannot be faked with a button on the attacker's website (they cannot discover the cookie.)
- An XMLHttpRequest from Javascript on the attacker's site will fail in Firefox etc or produce that weak-but-serious warning in Internet Explorer (refuse to talk to IE??)
- This reuses the passcode cookies developed for the GridHTTP protocol
  - Introduces lots of new positives too

GridSite solution (3)

- Since cookies rather than X.509 certificates are what really matters, can now login via other methods.
  - eg use Shibboleth username/password to access rights normally associated with an X.509 name
  - Use Kerberos to login as mcnab@hep.man.ac.uk rather than /C=UK/.../CN=andrew mcnab
- This integrates with GridSite 1.5.x support for non-X.509 credential types, and with existing GridHTTP

CSRF within the site

- However, if the attacker can get their Javascript onto the site, they can use XMLHttpRequest to trick the browser into doing other things elsewhere on the site.
- We can try restricting cookies to zones of the site with the path= option when they're issued.
  - But XMLHttpRequest can be used to get one of these cookies, as the browser still has the X.509 user cert etc.
- However, XMLHttpRequest doesn't work across sites...

Login sites

- I mentioned that GridSite's new architecture allows username/password login too, using a login page/script which creates the passcode and cookie.
- If we place this page on another virtual server within the same domain, can issue cookies but cannot be reached by XMLHttpRequest!
  - if https://www.gridpp.ac.uk/... is the site
  - then https://login.www.gridpp.ac.uk/... is the login site

Login sites (2)

- The site login.www.gridpp.ac.uk can issue cookies visible to any site within www.gridpp.ac.uk, but can issue path restrictions to limit what zones within that.
- This means we can separate the site into separate authorization zones and at least prevent escalation attacks from one to another.
- With X.509 user certs, it's still simple for the user: click “Go to login”, then “Login” button, then sent to original page.

Login page

CSRF etc - Andrew.McNab@manchester.ac.uk

MANCHESTER 1824

Summary

- XSS attacks are the developers fault!
- CSRF is developers fault if they don't use double-submit cookies; still vulnerable due to Microsoft Internet Explorer.
- Users must take security warning messages seriously.
- Users must be able to take security warnings seriously.
- GridSite has produced a general double-submit cookie framework for websites using X.509.