ENEE 457: Computer Systems Security
11/14/16

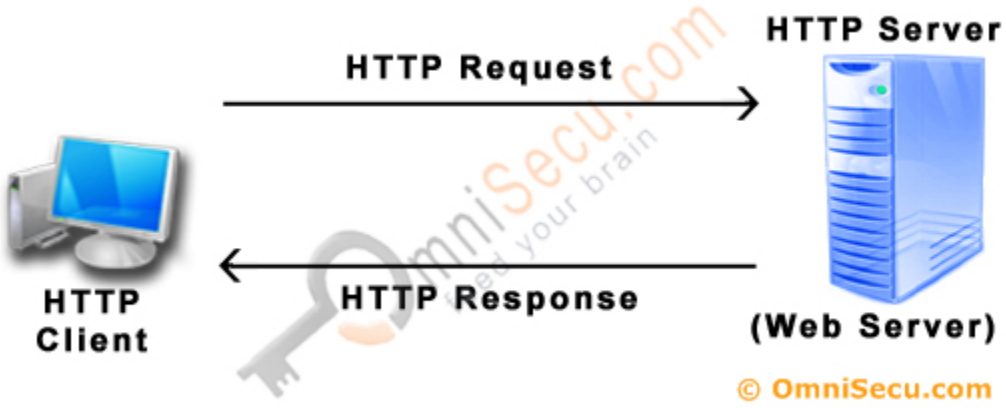
Lecture 20
Web Security

Charalampos (Babis) Papamanthou

Department of Electrical and Computer Engineering
University of Maryland, College Park

Back in the old days

- Clients will just send http requests
- Browser would just display information
- Servers will reply after talking to databases or other machines, sending mostly static content
- So the only attack vector was the server
  - We saw such attacks, e.g., buffer overflow



Now…

- Browser is super complicated
  - Java Script (you can execute Turing-Complete code at the client side)

www.w3schools.com/js/tryit.asp?filename=tryjs_intro_lightbulb

Run »

Result Size: 839 x 657

```
<!DOCTYPE html>
<html>
<body>

<h1>What Can JavaScript Do?</h1>

<p>JavaScript can change HTML attributes.</p>

<p>In this case JavaScript changes the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on the
light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the
light</button>

<p>Click the button to calculate x.</p>
    <button onclick="myFunction()">Try it</button>
    <br/>
    <br/>Enter first number:
    <input type="text" id="txt1" name="text1">Enter second number:
    <input type="text" id="txt2" name="text2">
    <p id="demo"></p>
    <script>
      function myFunction() {
        var y = document.getElementById("txt1").value;
        var z = document.getElementById("txt2").value;
        var x = y * z;
        document.getElementById("demo").innerHTML = x;
      }
    </script>

</body>
</html>
```

What Can JavaScript Do?

JavaScript can change HTML attributes.

In this case JavaScript changes the src (source) attribute of an image.

Turn on the light Turn off the light

Click the button to calculate x.

Try it

Enter first number: 4 Enter second number: 3

12

2013-05-DNSSEC-....pdf Show all

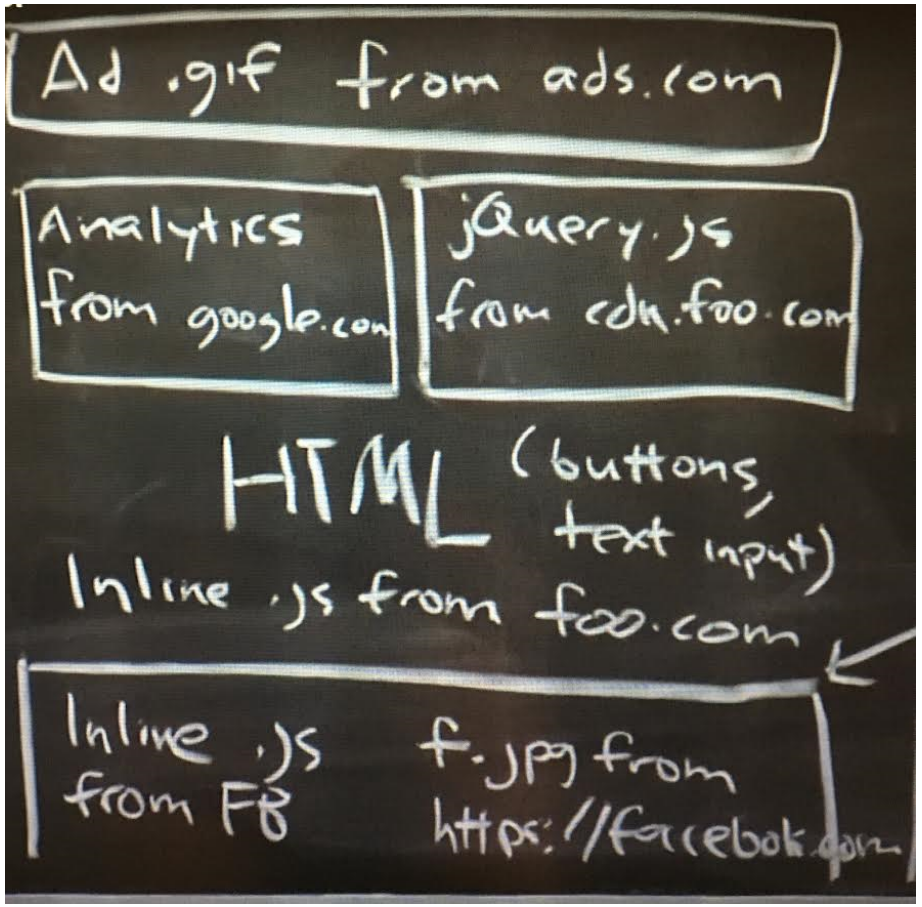
Search Windows ENG 8:15 PM 11/13/2016

Desktop applications VS Web Application

- Word comes from Microsoft
- Turbotax comes from the company that created
- But web applications?
  - Many different parties provide content
  - Think about cnn.com
  - You look at one page, but there are a lot of parties contributing content

Various Issues Here

- A web page can
  - Call JS libraries from different websites
  - Have inline JS
  - Load other html frames that have different inline JS
  - (the loaded frame can be https! while the loader frame can be http)
- Main problem: Can JS from different origin mix state?



Solution to that: Same Origin Policy

- **GOAL:** Two websites should not be able to tamper with each other
- But what does "non-tampering" mean? Hard.
  - One website should not be able to overwrite the display of another website without permission
  - But, two websites should be able to exchange data for collaboration
  - A website taking Google maps data and restaurant data
- **STRATEGY:** Each frame is assigned an origin and it can access resources only from its origin
- What is origin?
  - Protocol+hostname+port
  - http://foo.com/index.html (port 80)
  - https://foo.com/index.html (port 443)
  - http://bar.com: 8181/

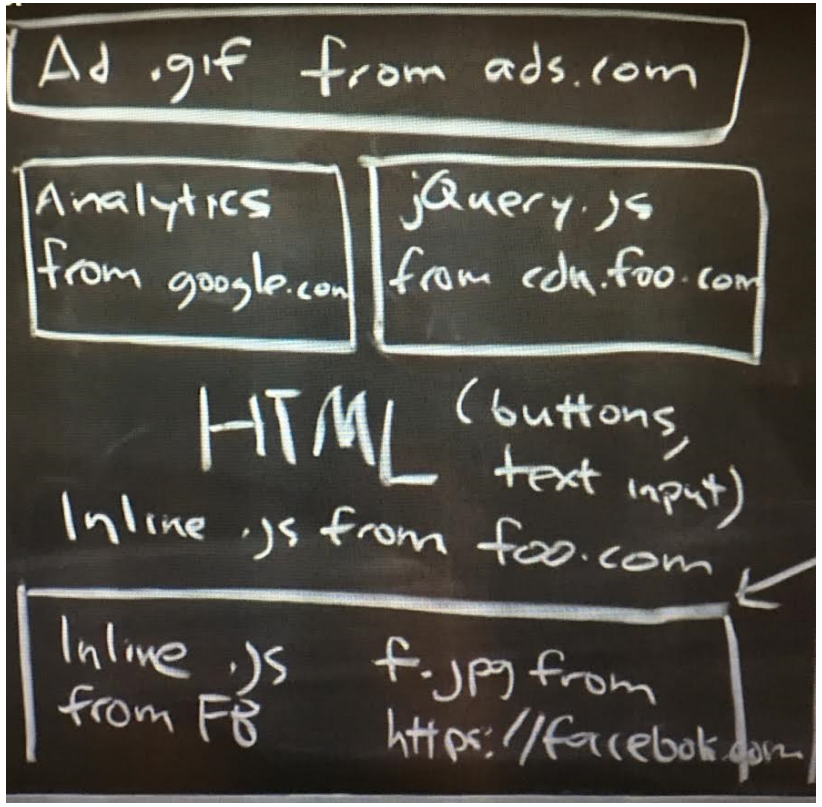
Four ideas behind SOP policy

- Each origin has client side resources
  - e.g., cookies for password remembering
  - DOM storage (basically a key-value store)

- Each frame gets the origin of its URL
- Scripts execute with the authority of its frame
  - If foo.com imports some javascript, then this would only be able to access cookies from foo.com
- Passive content does not get any authority

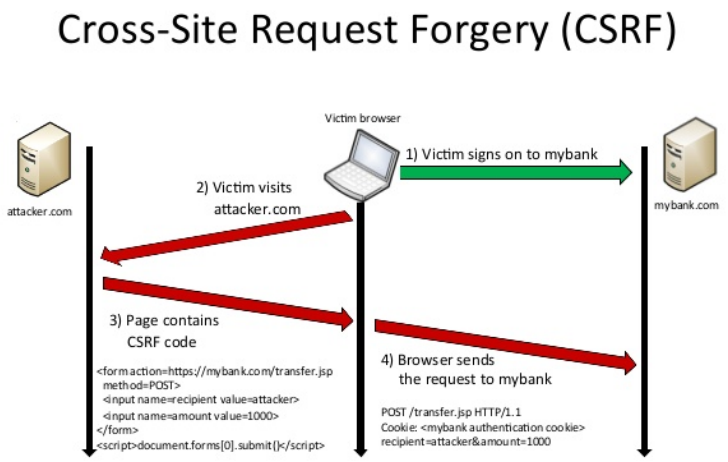
Let’s go back to our example

- Analytics and jQuery can access variables from foo.com
- But not from Facebook.com since they are in a different frame
- They can only communicate with a well-defined interface, POST-message (and foo.com can choose not to receive the message).



Cross-site request forgery

- Careful with cookies
  - I log into bank.com, and a cookie is planted in my machine with the password
  - Then the attacker makes me access a webpage, where he embeds
    - http://bank.com/xfer?amount=500 &to = attacker
  - Then my browser will read my cookies and it will do the transfer against my will…(CSRF)
  - How to prevent? Embed some randomness on form generation that the attacker cannot possibly know (apart from the cookie)



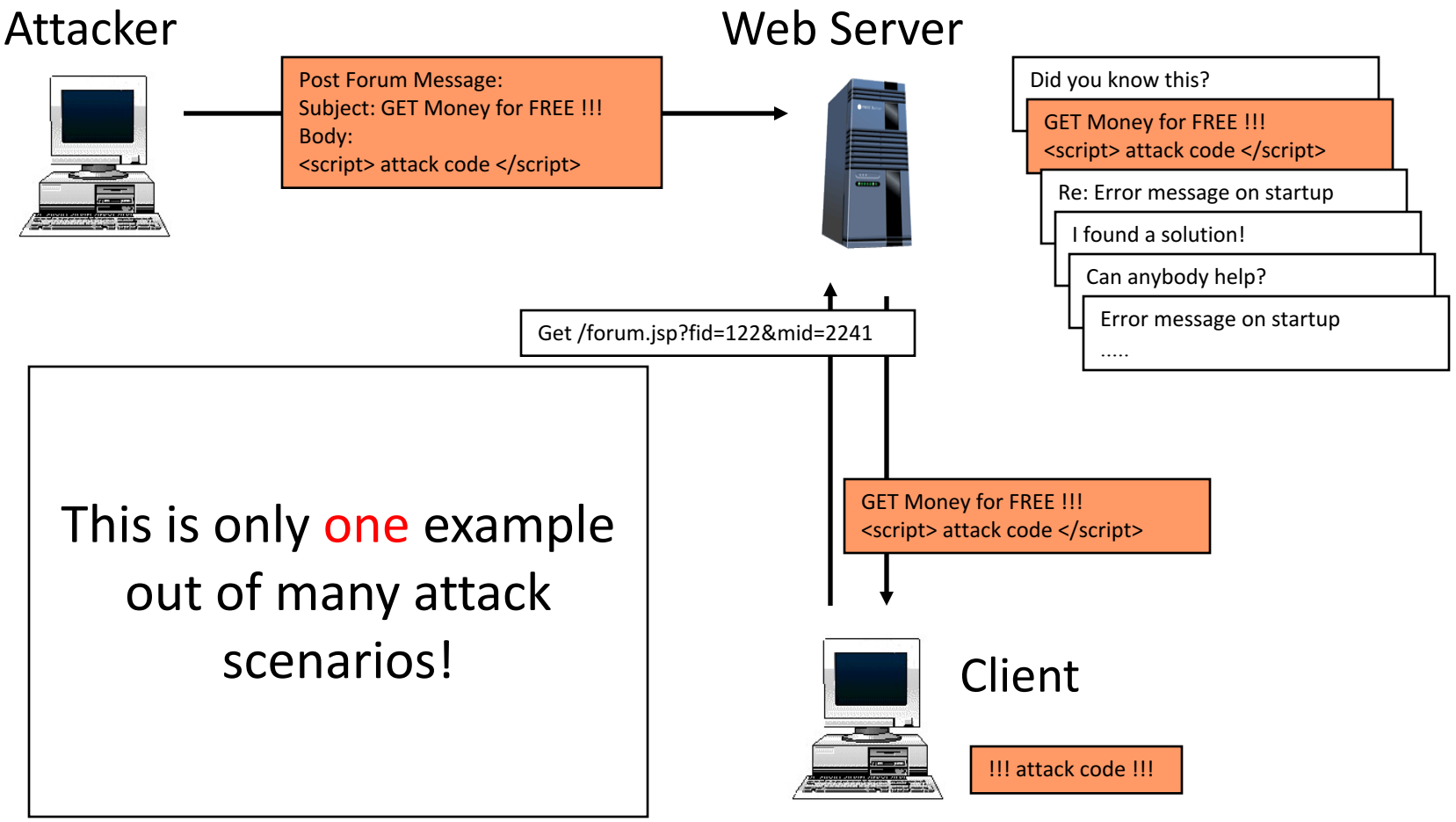
DNS rebinding attack

- Run attacker controlled JS with authority of victim
- That is possible only if victim call JS as a library, not easy for attacker
- So attacker gets the user to click on his webpage, and downloads its JS
- ATTACK:
  - Attacker registers attacker.com
  - User generates attacker.com DNS query
  - First it returns the correct IP, but with short TTL field (so it is not cached)
  - So at that point, attacker's code is at victim's machine, but with no right, due to SOP
  - The next DNS query attacker code does to attacker.com, the result is changed by the attacker, pointing to the victim address!
  - So SOP is broken, and now attacker's code can read cookies and access same domain!
  - TO solve: Ignore TTL and always cache the first request, this is called DNS pinning

Cross Site Scripting (XSS)

- Recall the basics
  - scripts embedded in web pages run in browsers
  - scripts can access cookies
    - get private information
  - scripts controlled by the same-origin policy
- Why would XSS occur
  - Web applications often take user inputs and use them as part of webpage (these inputs can have scripts)

XSS-Attack: General Overview



This is only one example out of many attack scenarios!

How XSS Works on Online Blog

- Everyone can post comments, which will be displayed to everyone who view the post
- Attacker posts a malicious comment that includes scripts (which reads local authentication credentials and sends to the attacker)
- Anyone who view the post can have local authentication cookies stolen
- Web apps will check that posts do not include scripts, but the check sometimes fail.
- Bug in the web application. Attack happens in browser.

XSS Example

- http://www.steve.org.uk/Security/XSS/Tutorial/simple.html

Protection against XSS attacks

- Sanitize the input
- Make sure it does not contain any scripts!

SQL Injection Attack

- Many web applications take user input from a form
- Often this user input is used literally in the construction of a SQL query submitted to a database. For example:

```
SELECT user FROM table
  WHERE name = 'user_input';
```

- An SQL injection attack involves placing SQL statements in the user input

SQL: Standard Query Language

- SQL lets you access and manage (Query) databases
- A database is a large collection of data organized in tables for rapid search and retrieval, with fields and columns

A field or Column

A Record or Row

First_Name	Last_Name	Age
John	Myers	19
Maria	Palm	22
Alex	Klein	21
.....	....	....

Table: ENEE-459

SQL Syntax

```
SELECT  First_Name
FROM ENEE-459
WHERE age=21
```

- SELECT statement is used to select data FROM one or more tables in a database
- Result-set is stored in a result table
- WHERE clause is used to filter records

SQL Syntax

```
SELECT  Last_Name
FROM ENEE-459
WHERE age=21
ORDER BY First_Name ASC
LIMIT 3
```

- ORDER BY is used to order data following one or more fields (columns)
- LIMIT allows to retrieve just a certain numbers of records (rows)

Login Authentication Query

- Standard query to authenticate users:

  select * from users where user='$usern' AND pwd='$password'

- Classic SQL injection attacks
  - Server side code sets variables $username and $passwd from user input to web form
  - Variables passed to SQL query

  select * from users where user='$username' AND pwd='$passwd'

- Special strings can be entered by attacker

  select * from users where user=M' OR '1=1 AND pwd=M' OR '1=1

- Result: access obtained without password
- Solution: Careful with single quote characters
- Filter them out!