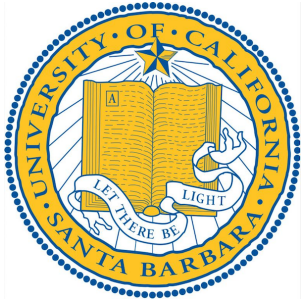
CS 290B

Scalable Internet Services

Bryce Boe

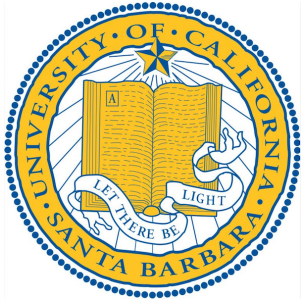
November 13, 2014



Today's Agenda

Web Security Basics

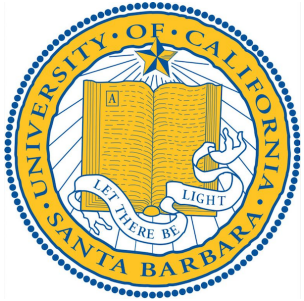
- HTTPS
- Firewalls
- SQL Injection
- Cross-Site Scripting
- Cross-Site Request Forgery



A Brief History of HTTPS

Protocol	Year
SSL 1.0	n/a
SSL 2.0	1995
SSL 3.0	1996
TLS 1.0	1999
TLS 1.1	2006
TLS 1.2	2008
TLS 1.3	TBD

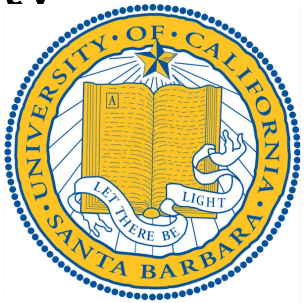
- Originally developed by Netscape in 1995.
- HTTPS: “HTTP Secure”
  - HTTP layered on top of TLS
- TLS: “Transport Layer Security”
  - Started life as SSL “Secure Sockets Layer”
  - When standardized it became renamed TLS
- TLS is not just for HTTP and is a general purpose security layer.



HTTPS - Goals

What do we want from a secure sockets layer?

- **Encryption**
  - The data that is sent can not be read by intermediaries
- **Authentication**
  - I can be sure I am speaking to the intended partv
- **Integrity**
  - Our communications can not be tampered with.



HTTPS

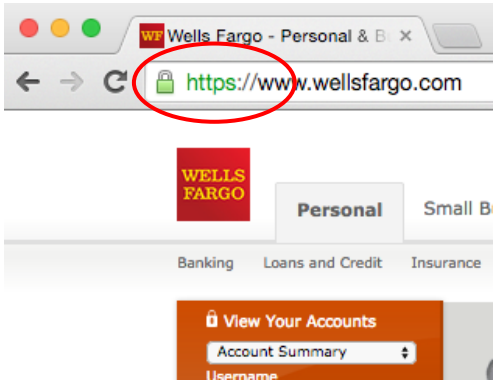
How does TLS do this?

- **Certificates** are used to verify the identity of the server (and more rarely, the client)
- **Asymmetric cryptography** is used to establish a shared key.
- Once a shared key has been established, **symmetric cryptography** is used for the session.

TLS allows combinations of different cipher suites to be used.

HTTPS - Certificates

You navigate your browser to https://wellsfargo.com



- Your browser wants to be able to tell you if the party you are communicating with is legitimate.
  - HTTP on its own won't protect against man-in-the-middle attacks
- A private key can authenticate a party, but it's infeasible for your browser to know about all secure websites directly.
  - The list of private keys would be in the millions and would be dynamic

HTTPS - Certificates

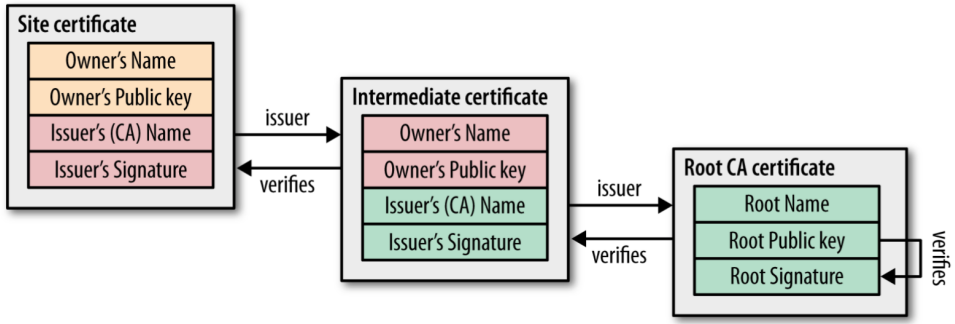
Insight: we can use private key signatures to transitively authenticate someone. Example:

- Alice trusts Bob and has his public key
- Bob knows Charlie and has his public key
- Charlie wants to talk to Alice and presents his public key
- Alice doesn't know if this is really Charlie, or someone else who has handed over their own private key.
- In order to solve this, Charlie has Bob write down "Charlie's public key is [...]" and sign it with his private key.
- Because Alice trusts Bob, she knows Charlie is legitimate when he provides the Bob-signed document.
- **This document is known as a certificate.**

HTTPS - Certificates

Solution: Browser maintains a small list of trusted Certificate Authorities (CAs)

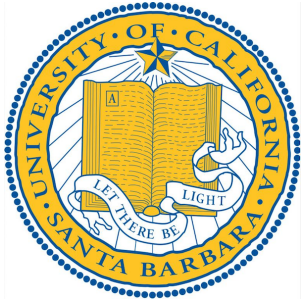
- Any website can present a certificate issued by a CA and the browser can trust that the party is legitimate
- Certificates can be chained.
  - “Root” CA vs. “intermediate” CA



HTTPS - Certificates

Certificates can be revoked.

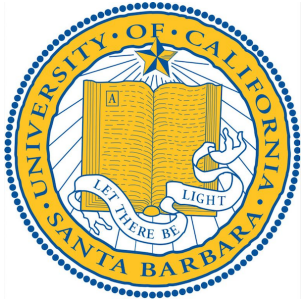
- Why might we need to revoke a certificate?
  - Private key compromised
  - Intermediate CA was compromised
- Two main mechanisms for certificate revocation
  - Certificate Revocation List
    - Periodically get a list of all revoked certificates. If a connection is attempted with a revoked certificate, do not allow it.
  - Online Certificate Status Protocol (OCSP)
    - At request time, query the CA to check if revoked.
  - Advantages of each?



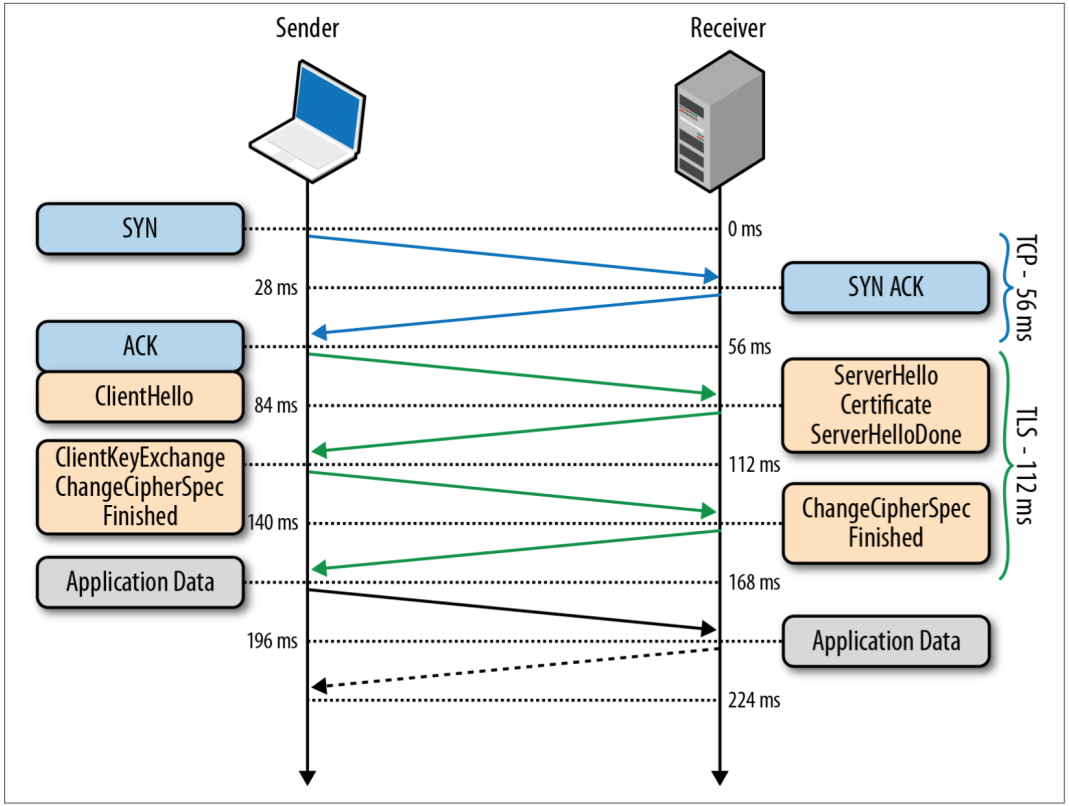
HTTPS - Key Exchange

Once we are confident that the other party is legitimate, we want to establish a shared session key

- Two main techniques in use today. Details of each are outside the scope of this course.
  - RSA
    - Developed by Rivest, Shamir and Adleman in 1977
    - Based on the difficulty of integer factorization
  - Diffie-Hellman
    - Developed by Whitfield Diffie & Martin Hellman in 1976
    - Based on the difficulty of the discrete logarithm problem
  - Fun fact: both techniques were invented years earlier by British signals intelligence (GCHQ).



HTTPS - SSL Handshake



After TCP setup:

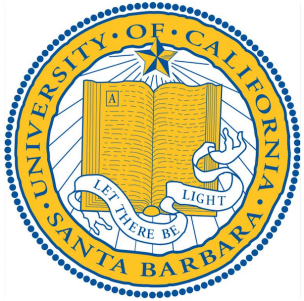
1. Client initiates SSL handshake with a list of CipherSuites and a random number
2. Server responds with its cert, selected CipherSuite and a random number
3. Client responds with more randomness, encrypted with server's public key
4. Each side computes session key based on the three random numbers.
5. Both sides switch to using symmetric key.

"High Performance Browser Networking", Page 51

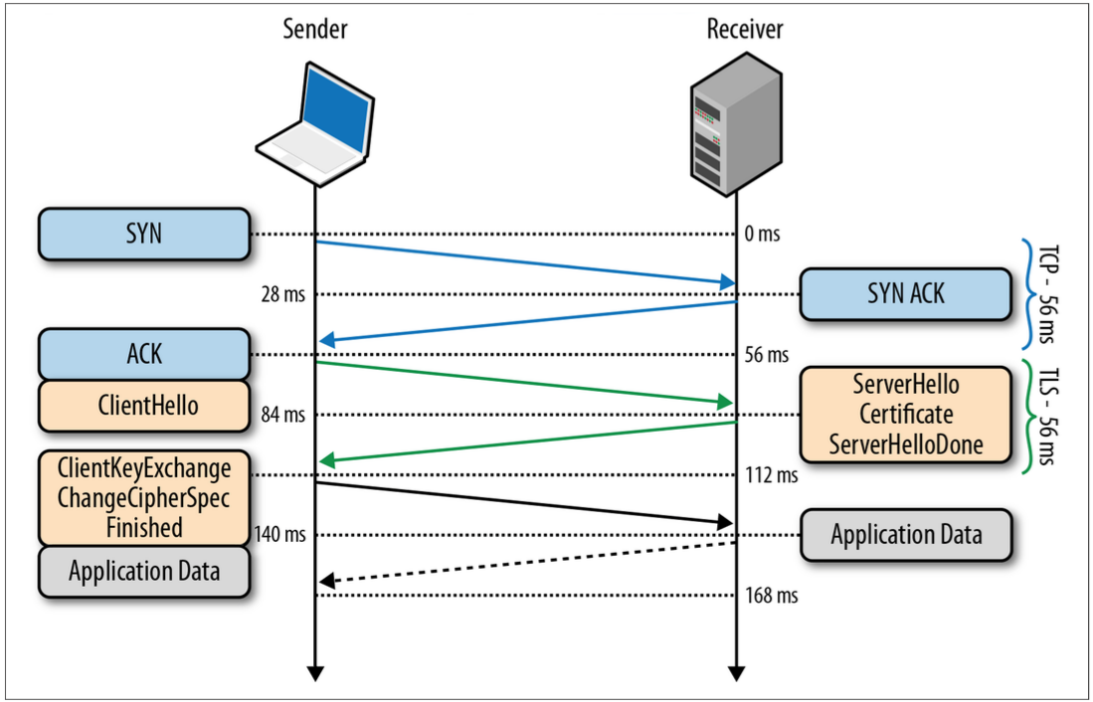
HTTPS - SSL Handshake

Latency involved in two extra round trips is expensive.

For repeated access by a client we can speed this up...



HTTPS - Abbreviated SSL Handshake



After TCP setup:

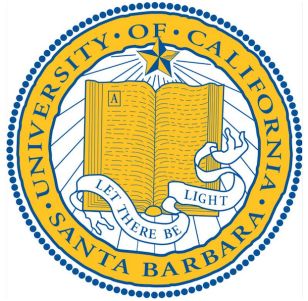
1. Session ID is added to connections with new hosts
2. Client initiates SSL handshake with previously-used Session ID
3. Server acknowledges previously used Session ID
4. Each side computes session key based on the remembered random numbers.
5. Both sides switch to using symmetric key.

High Performance Browser Networking", Page 57

HTTPS - Goals

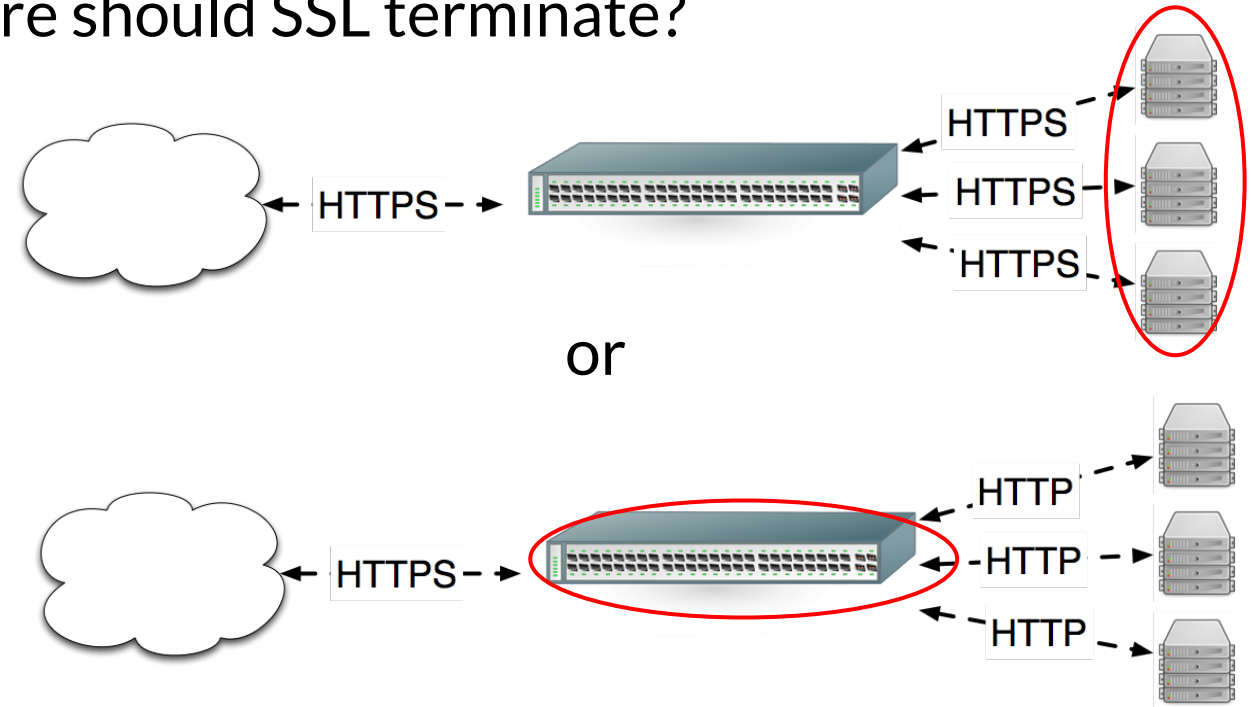
Have we achieved our goals?

- **Encryption**
  - Because no other party can know the three random numbers with which we generated the session key, our data can not be read by intermediaries
- **Authentication**
  - Because the public key used by the server has an associated certificate, and because I trust the CA, I know this is the intended party
- **Integrity**
  - Each TLS frame includes a Message Authentication Code (MAC) that prevents tampering.



HTTPS

Where should SSL terminate?



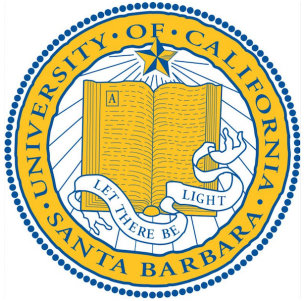
HTTPS Termination

Advantages of terminating SSL at the load balancer

- If each app server maintained its own cache, it would frequently miss.
- Load balancer can be built with hardware acceleration for TLS handshake.
- Load balancer can see inside each request, potentially improving its balancing decisions.
- Private key just sits in one place

Advantages of terminating SSL at the App Server

- Data between load balancer and app servers stays private



Firewalls

Used to secure devices from outside access

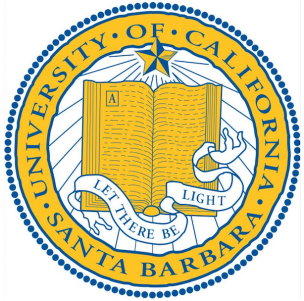
- Enforces access control policy between two networks
- Two designs: restrict “bad traffic” or permit “good traffic”
- Designed to operate at different layers of the network stack

Can influence traffic going out as well

- e.g. “Great Firewall of China”
- We are primarily concerned with inbound traffic

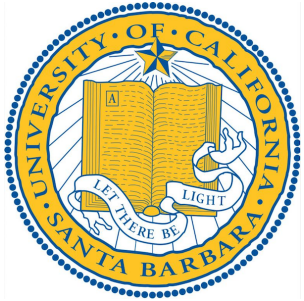
Can be standalone hardware devices or software

- Often included in multi-purpose device e.g. switch or load balancer



When to use a Firewall

- Use firewalls only when they significantly reduce risk
- Employ firewalls to protect sensitive data
  - Critical PII, PCI compliance, etc.
- Firewalls should be treated like perimeter security
  - Like the locks on your house
  - You should eliminate “extra firewalls”
- Consider the value of what you are protecting and the cost to firewall it
  - Like your house, there are some things that are not worth protecting
    - Can you think of examples?



When to use a Firewall

Firewalls are often overused

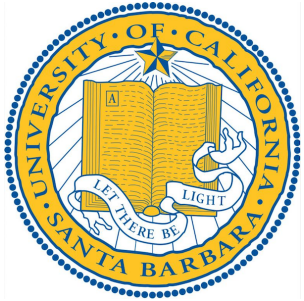
“Failed firewalls are the #2 driver of site downtime after failed databases”

- **Scalability Rules**, by Martin Abbott

Can create difficult to scale chokepoint for either network traffic or transaction volume

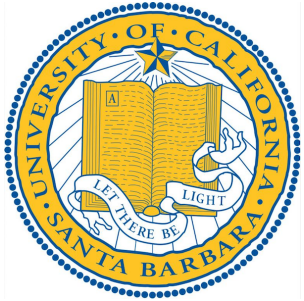
May have impact on availability

- DDoS attacks on session state memory



Common Firewalls

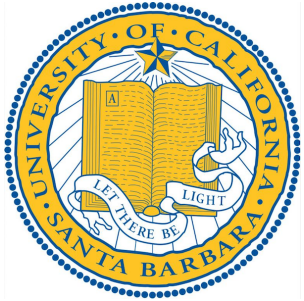
- Software
  - Included with operating systems (ipfw)
  - Can also buy standalone
- Hardware
  - Cisco ASA, Citrix AppFirewall, F5 AFM
- EC2 Security Groups
  - Allow specific protocols and ports to access server
  - Can restrict machines to only accept traffic from Elastic Load Balancer
- A typical large scale web service will use both hardware and software firewalls



Three common attacks

Next, we will look at three common security vulnerabilities on the web, and how to mitigate them:

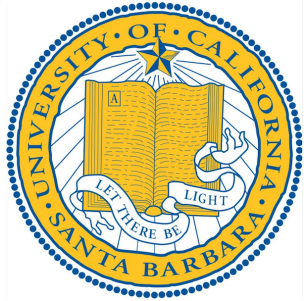
- **SQL injection**
  - Getting a database to execute bad SQL
- **Cross-site scripting**
  - Getting a browser to execute bad javascript
- **Cross-site Request Forgery**
  - Getting a browser to submit bad data



SQL Injection

What's wrong with this code?

```
def create
  user_id = params['user_id']
  comment_text = params['comment_text']
  sql = <<-SQL
    INSERT INTO comments
    user_id=#{user_id},
    comment=#{comment_text}
  SQL
  ActiveRecord::Base.connection.execute(sql)
end
```

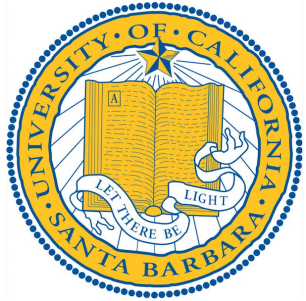


SQL Injection

```
def create
  user_id = params[‘user_id’]
  comment_text = params[‘comment_text’]
  sql = <<-SQL
    INSERT INTO comments
    user_id=#{user_id},
    comment=’#{comment_text}’
  SQL
  AR::Base.connection.execute(sql)
end
```

What if a user submitted the following parameters?

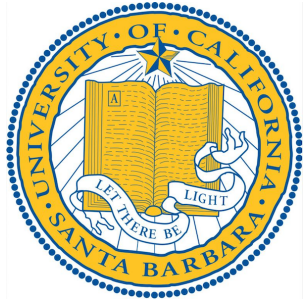
```
user_id=5
comment_text=’; UPDATE users SET
admin=1 where user_id=5 and ‘1’=’1
```



SQL Injection

```
sql = <<-SQL
  INSERT INTO comments
  user_id=#{user_id},
  comment='#{comment_text}'
SQL
```

```
INSERT INTO comments
    user_id=#{user_id},
    comment=''; UPDATE users SET
admin=1 where user_id=5 and '1'='1'
```

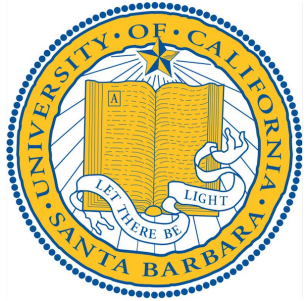


SQL Injection

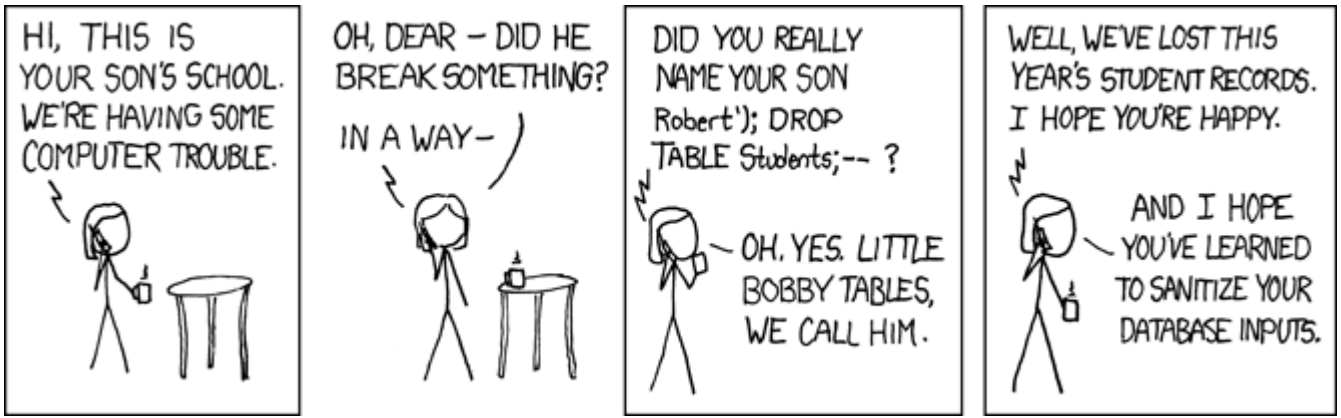
How do we mitigate this?

- Never insert user input directly into a SQL statement without sanitizing it first.
- Rails does a lot of the work here for you:
  - Access through the AR ORM is safe
  - If you need to sanitize custom sql, you can use the sanitize method:

```
SELECT *
FROM comments
WHERE id=#{Comment.sanitize(params[:id]))}
```



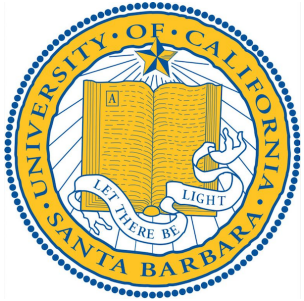
SQL Injection



Cross-site Scripting

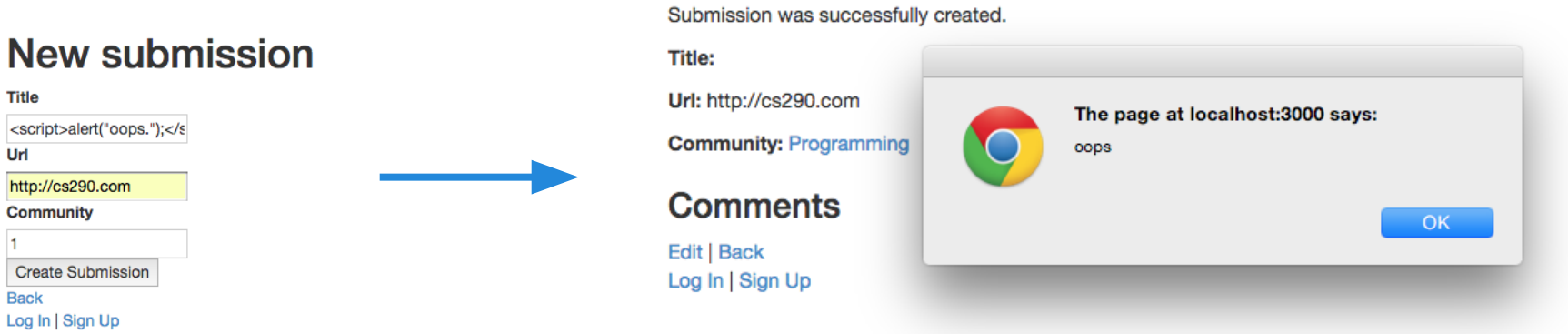
If I can get another user to execute arbitrary javascript in the context of another page, it can be quite dangerous:

- Ajax requests are limited to the domain that the JS originated from, but requests back to that domain will include all relevant cookies
- For example, if I can execute arbitrary JavaScript in the browser that is running http://wellsfargo.com, all Ajax requests will occur with the user's current cookies (and session)
- Cross-site scripting is when one user can submit data that is displayed on another users screen and, when containing JS, gets executed.



Cross-site Scripting

Example:

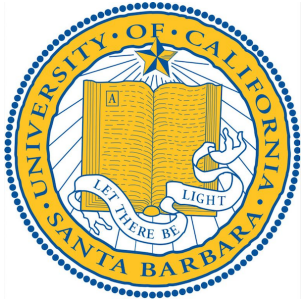


Cross-site Scripting

Because the application developer is just redisplaying the data submitted by one user into the DOM of another user, one user can do anything that the other, logged-in user can do.

- In an email website, it could send emails on your behalf, or read your mail
- On a bank website, it could potentially transfer money to an attacker

How might we prevent this?



Cross-site Scripting

Sanitize the data that you are displaying to the user.

Rails does this for you. Example:

If I enter form data as:

```
<script>alert(“oops”);</script>
```

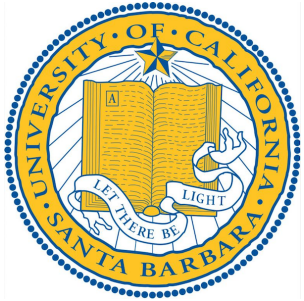
Rails will save it to the database as exactly that.

But when I go to display it to the user:

```
<%= submission.title %>
```

Rails takes the title (exactly as above) and automatically converts it to

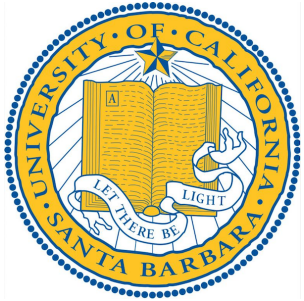
```
&lt;script&gt;alert(&quot;oops.&quot;);&lt;/script&gt;
```



SQL Injection & XSS prevention

How do we make sure we aren't making injection errors?

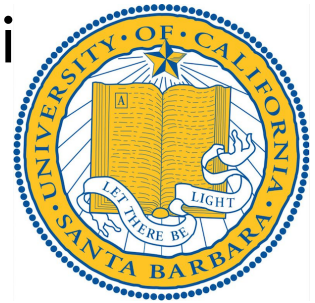
- Fuzzing: provide semi-random input to the application and watch for errors
- Tarantula: An automated tester that crawls your application and fuzzes for injection errors:
  - https://github.com/relevance/tarantula



Cross-site Request Forgery

Ajax is limited to accessing URLs from the domain it originated from, non-Ajax attacks are also possible.

- I can set up a form on domain1.com to POST to domain2.com.
- Alternately, I can use an img tag to GET a resource that has side-effects
- I can't do it with Ajax, so I can't see the result, but i still do damage.
- Like with XSS, All session cookies, etc. will be transmitted to the server



Cross-site Request Forgery

`https://labapp.com`

`https://labapp.com/comments/1`

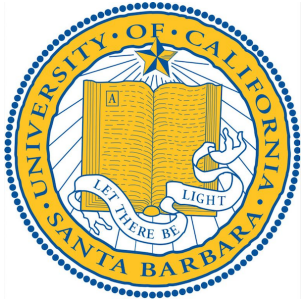
Comment:

Check this image out:

```
<img src="http://bank.example.com/withdraw?account=Alice&amp;amount=1000000&amp;for=Mallory">
```

Submit

When viewed, the image tag attempts to load, performing `HTTP GET` *with session cookies*



Cross-site Request Forgery

`https://cs290.com`

Sign up for a fun service!

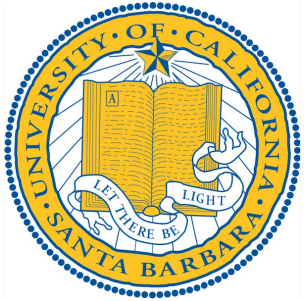
Email:

Submit

Actually submits to

`https://wellsfargo.com/txn/new`

Your bank transfer is complete!



Cross-site Request Forgery

How do we mitigate this?

- First, make proper use of HTTP semantics. GETs shouldn't have side effects.
- For form POSTing, a common technique is to add a random token as a hidden field in each form rendered.
  - If the form is submitted without the token, the request is denied.

Cross-site Request Forgery

How does this look in Rails?

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  protect_from_forgery with: :exception
end
```

This causes a hidden form field to silently be added to all forms:

```
<input name="authenticity_token" type="hidden" value="
SLNTcHBDnzl21gPHVoF3DAUEGZLAxWYqZ1FQxBBlmek=">
```