LECTURE 30

Application Security Basics

Your app is about to meet the Internet — here's how to keep it, and its users, safe.

CSC336 Web Technologies

OVERVIEW

In This Lecture

1. The CIA triad — confidentiality, integrity, availability
2. Two guiding principles: least privilege and defence in depth
3. HTTPS/TLS, certificates, and security-related response headers
4. Authentication vs. authorization, revisited
5. Input validation, output encoding, and secrets management

WHY IT MATTERS

Security for a Class Project?

1. "Security is for big companies, not my class project" — this is a mistake
2. The moment an app is reachable on the Internet, automated bots probe it 24/7 for common weaknesses
3. Security is not a bolt-on at the end — it's a way of thinking throughout development

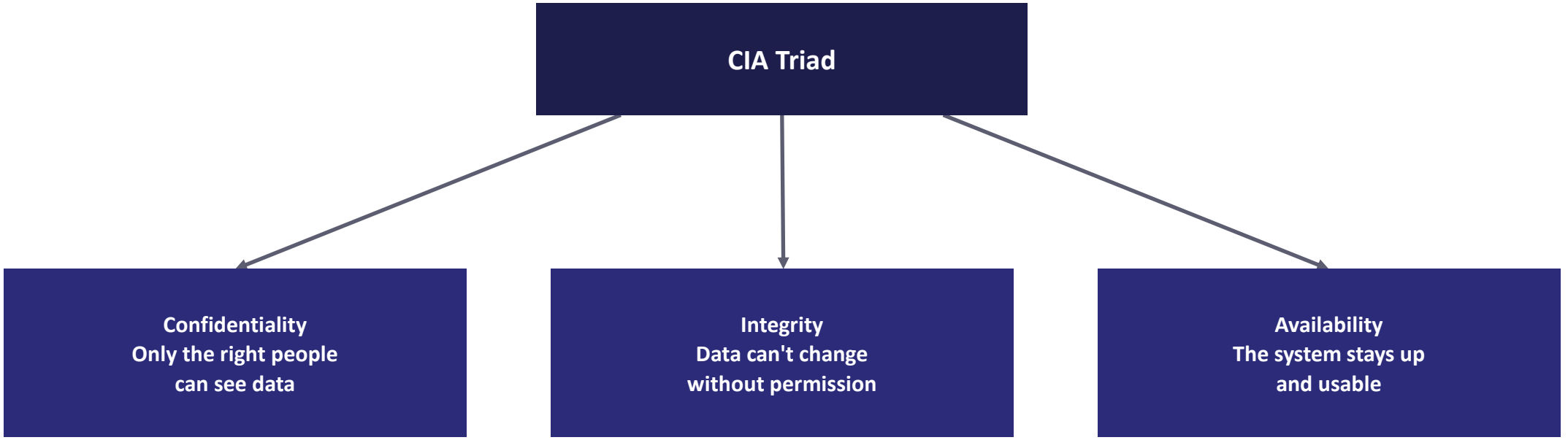
WHY IT MATTERS

Security Is a Process, Not a Checklist

NOTE

You cannot make an application "100% secure." Security is about reducing risk to an acceptable level and reacting quickly when something goes wrong — not reaching some finish line where you are "done."

Three Goals of a Secure System



Nothing to do with the intelligence agency — every security decision maps back to protecting one (or more) of these three properties.

What Each Property Actually Means

CONFIDENTIALITY

- Information is only visible to people supposed to see it
- A password database should not be readable by an outside attacker
- Main tools: encryption and access control

INTEGRITY

- Data can't be changed without authorization — and changes are detectable
- Nobody quietly edits a stored grade without a trace
- Main tools: checksums, digital signatures, access control

AVAILABILITY

- The system stays up and usable for legitimate users
- A denial-of-service attack floods a server to knock it offline
- Attacks availability without ever reading or changing data

Give Only the Access That's Needed

1. Every user, process, or piece of code should have ONLY the permissions it strictly needs — nothing more
2. Example: if your API only reads/writes one collection, its DB user shouldn't be able to drop the whole database
3. If an attacker compromises that account, least privilege limits how much damage they can do

It Applies at Every Level

Level	What Least Privilege Looks Like
Operating system	A web server process should not run as root / Administrator
Database	The app's DB user gets only the read/write it actually needs
Application	A regular user can't reach admin-only actions; a read-only key can't write
Team	Only developers who genuinely need production DB credentials get them

The Question to Always Ask

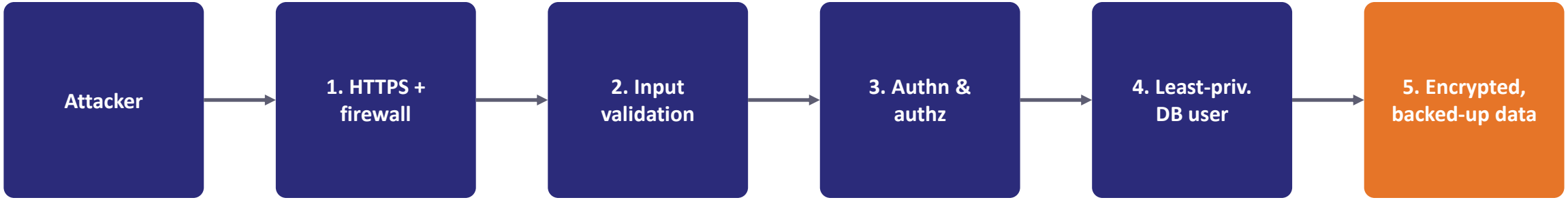
TIP

"What is the SMALLEST set of permissions this needs to do its job?" Grant that, not more. It is much easier to add a permission later than to clean up after a breach caused by over-broad access.

Never Rely on One Layer

1. Layer multiple, independent defences so that if one fails, others are still standing
2. Like a castle: not just a tall wall — also a moat, a drawbridge, guards, an inner keep
3. If authentication alone had a bug, one flaw would expose your entire system

Six Layers in a Web Application



With only step 3 (authentication) and nothing else, one login bug could expose everything. With defence in depth, one failure does not mean total compromise.

Plain HTTP Is Not Safe

1. HTTP sends data as plain text — anyone intercepting traffic (shared Wi-Fi, your ISP) can read it, including passwords
2. HTTPS = HTTP layered on top of TLS (Transport Layer Security), which encrypts the connection
3. TLS is the modern successor to SSL — people still say "SSL" out of habit, though SSL itself is obsolete

What HTTPS Gives You — Mapped to CIA

HTTPS Provides	CIA Property Protected
Encryption — nobody eavesdropping can read the traffic	Confidentiality
Integrity — data can't be silently modified in transit	Integrity
Authentication of the server via a certificate	(Confirms identity, backs both)

Certificates and Certificate Authorities

1. A TLS certificate is a digital document proving a server's identity + its public key
2. Issued by trusted organizations called Certificate Authorities (CAs)
3. If a certificate is missing, expired, or doesn't match the domain, the browser warns instead of loading the page

The TLS Handshake

The server obtained its certificate from a CA in advance — the browser only has to verify it, not request one on the spot.

HTTPS Got Easy

TIP

Free, automated certificate services like Let's Encrypt made HTTPS free and easy, which is why almost the entire web uses it by default. Most hosting platforms (Lecture 32) issue and renew certificates for you automatically.

Security-Related HTTP Response Headers

Header	What It Does
Strict-Transport-Security	"Always use HTTPS for this site" — even if the user types http://
X-Content-Type-Options: nosniff	Stops the browser guessing a file's type — blocks disguised malicious files
X-Frame-Options	Controls whether your page can load inside an <iframe> — prevents clickjacking
Content-Security-Policy	Restricts what sources of scripts/styles/resources a page can load

A Shortcut: the helmet Middleware

```
const express = require("express");
const helmet = require("helmet");

const app = express();
app.use(helmet()); // sets several security headers with good defaults
```

Two Different Questions

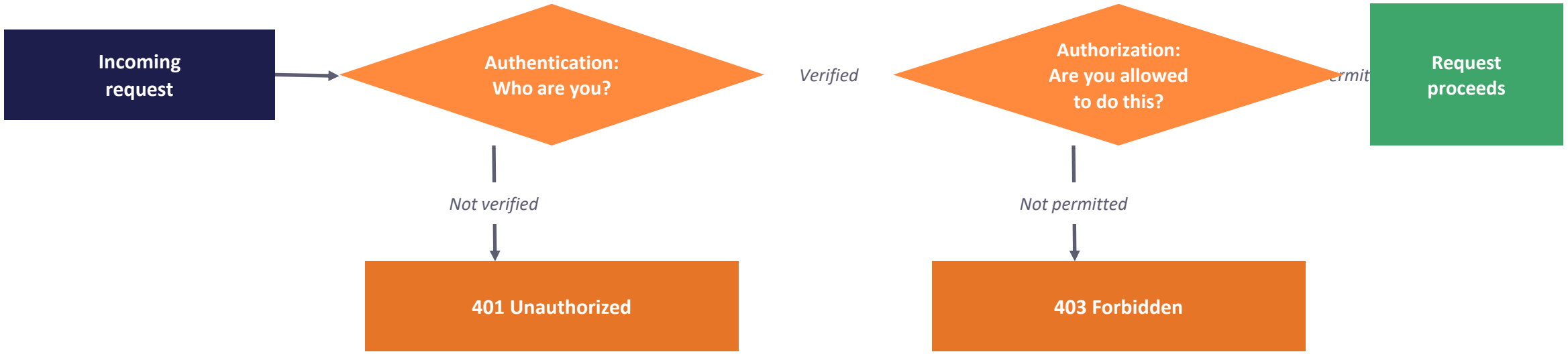
AUTHENTICATION

- "Who are you?"
- Verifying an identity — a password, a token, a proof
- Like showing your student ID at the university gate

AUTHORIZATION

- "What are you allowed to do?"
- Happens AFTER authentication — decides if this action is permitted
- Like your ID having (or not having) access to a specific lab

Every Request Answers Both Questions



A very common bug: checking authentication but forgetting authorization — e.g. GET /api/orders/:id rejects anonymous users, but returns ANY order to ANY logged-in user.

Two Checks, Not One

WARNING

This exact mistake — authenticated but not authorized — is common enough to have its own name: broken access control, covered in Lecture 31.

401 Unauthorized really means "you haven't proven who you are" (an authentication failure). 403 Forbidden means "I know who you are, but you're not allowed to do this" (an authorization failure). Always ask both questions.

Never Trust Data Coming In

1. Check that data from a form, URL parameter, upload, or request body matches what you expect — before using it
2. Never trust data just because it came from your own frontend
3. An attacker can send requests straight to your API with curl or Postman, bypassing your React forms entirely

Basic Server-Side Validation

```
app.post("/api/signup", (req, res) => {
  const { email, age } = req.body;

  if (typeof email !== "string" || !email.includes("@")) {
    return res.status(400).json({ error: "A valid email is required." });
  }

  const numericAge = Number(age);
  if (!Number.isInteger(numericAge) || numericAge < 13 || numericAge > 120) {
    return res.status(400).json({ error: "Age must be 13-120." });
  }

  res.status(201).json({ message: "Account created." });
});
```

Libraries like joi, zod, or express-validator define these rules once instead of writing manual if checks per field.

Client-Side Validation Is Convenience, Not Security

WARNING

React form validation gives instant feedback with zero network round trip — but it runs on the attacker's own computer, where they can disable or bypass it entirely. Always re-validate on the server too. Server-side validation is the real security boundary.

The Mirror Image of Input Validation

1. Transform data right before displaying/using it, so it can't be misread as code in that new context

2. If a user's name is literally <script>alert('hi')</script> and you insert it into HTML unencoded, the browser runs it — this is Cross-Site Scripting (Lecture 31)

3. React encodes {userName} in JSX automatically — the danger is deliberately bypassing that with dangerouslySetInnerHTML

What Counts as a Secret

1. Any information that must stay private for your system to remain secure
2. Database passwords, third-party API keys, JWT signing keys, cloud credentials — all secrets
3. Once pushed to a public Git repo, a secret is permanently compromised — even if you delete it later, it's still in the history

Never Commit Secrets to Git

WARNING

One of the most common and damaging mistakes: committing a .env file or hard-coded API key, especially to a public GitHub repo. Automated bots scan public repos for exposed keys within minutes of a push. If one leaks, rotate it — generate a brand-new secret and revoke the old one.

The Standard Practice

```
# .gitignore
node_modules/
.env

# .env.example  (safe to commit — no real secrets)
DATABASE_URL=your-mongodb-connection-string-here
JWT_SECRET=your-jwt-signing-secret-here
STRIPE_API_KEY=your-stripe-key-here
```

Commit .env.example so teammates know what to set up. On your hosting platform (Lecture 32), set real values through its own environment variable settings — never by uploading a .env file.

KEY TAKEAWAYS

Lecture 30 in Six Points

1. The CIA triad — Confidentiality, Integrity, Availability — describes the three goals every security control protects.
2. Least privilege: give every account, process, and piece of code only the permissions it strictly needs.
3. Defence in depth: layer multiple independent controls so one failure doesn't mean total compromise.
4. HTTPS/TLS encrypts traffic and lets the browser verify a server's identity via a CA-issued certificate.
5. Authentication ("who are you?") and authorization ("what are you allowed to do?") are distinct — check both, every time.
6. Validate input on the server, encode output on display, and never commit secrets to Git.