

SOFTWARE VULNERABILITY & SECURITY

AGENDA

FIRST HOUR LECTURE



1. *Security Vulnerability (Problem)*



2. *Common Web Attacks (Problem)*



3. *DevSecOps: How to Integrate Security in DevOps workflow? (Solution)*

SECOND HOUR HANDS-ON



Hands-On: A Unified Security Risk Evaluation Framework (Solution)

SECURITY



VULNERABILITY

Why worry?

World's Biggest Data Breaches & Hacks

Selected events over 30,000 records stolen

UPDATED: Jun 2024

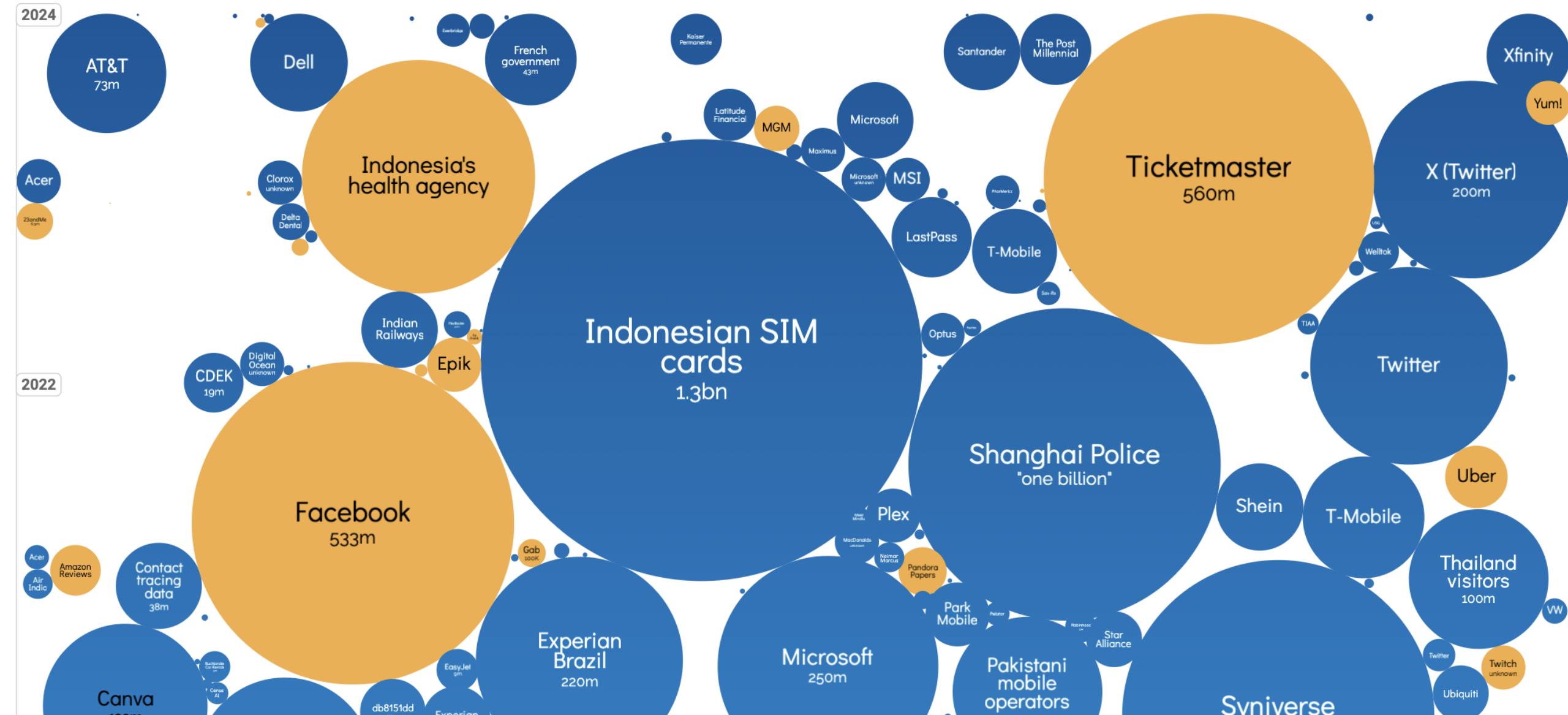
interesting story

size: records lost

filter

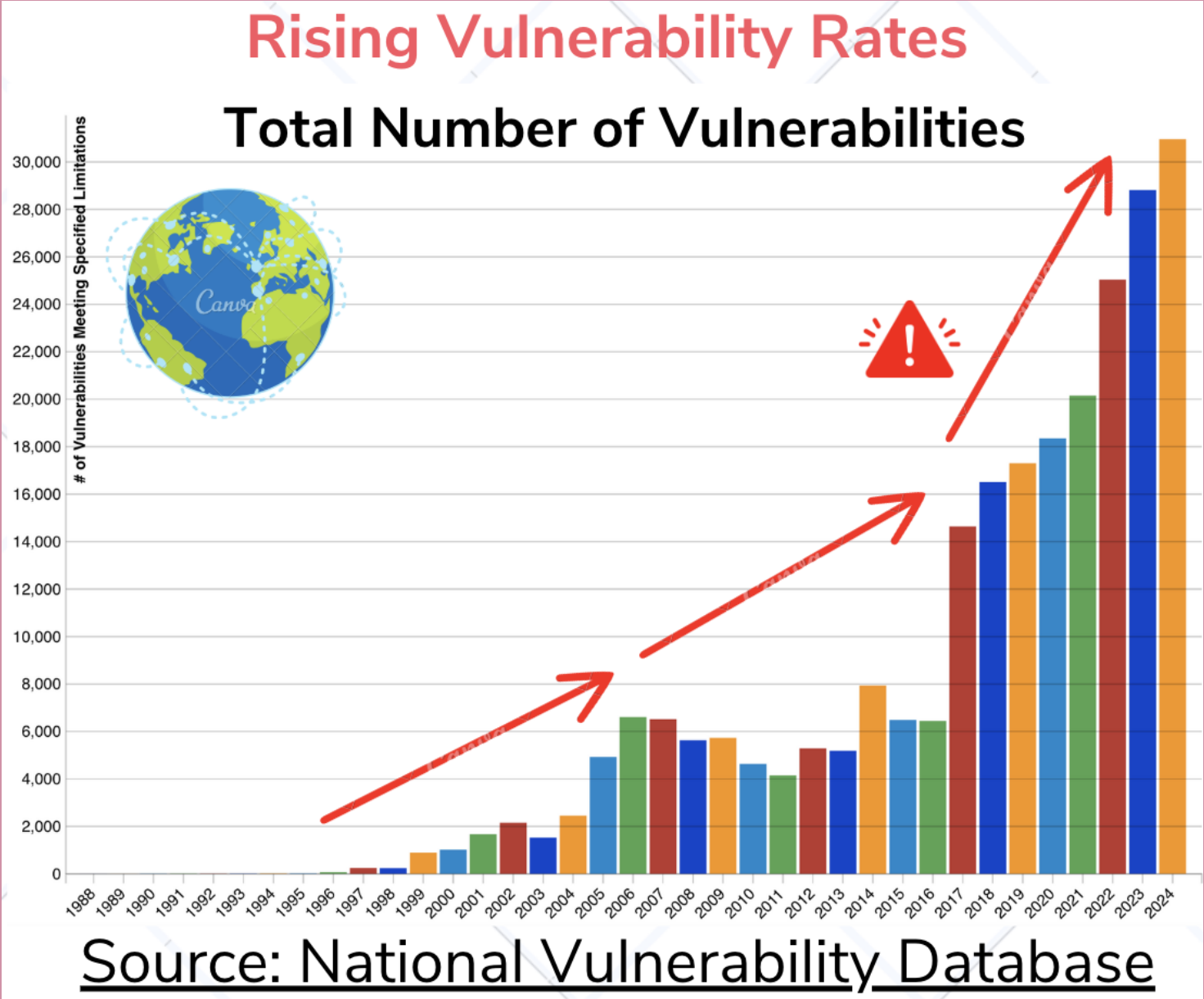
Source: <https://informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/>

search...





RISING SECURITY VULNERABILITIES: A GROWING CONCERN



In Australia...

major software companies and governments take action on cybersecurity



Microsoft pledges \$5 billion for Australia's digital infrastructure, skills, and cybersecurity. (October, 2023)



Google is investing \$1 billion in Australia to support digital infrastructure and cybersecurity.

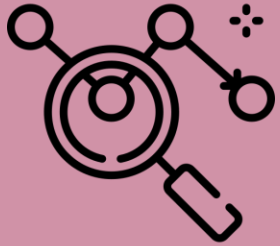


Australian government proposes 2023–2030 Australian Cybersecurity Strategy



Software Vulnerabilities Remain a Major Engineering Hurdle

Understanding CVE, CWE, NVD, and CVSS



Cybersecurity relies on identifying and tracking software vulnerabilities.

Four core components to support this process...



CVE
Common
Vulnerabilities
and Exposures



CWE
Common
Weakness
and Enumeration



NVD
National
Vulnerability
and Database



CVSS
Common
Vulnerability
Scoring System

CVE – Common Vulnerabilities and Exposures

- A list of publicly disclosed cybersecurity vulnerabilities.
- Managed by MITRE Corporation.
- Each CVE has a unique ID (e.g., CVE-2024-12345).
- Helps standardize naming and tracking of vulnerabilities.

CVE-2024-27193 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') vulnerability in PayU PayU India allows Reflected XSS.This issue affects PayU India: from n/a through 3.8.2. Published: March 15, 2024; 9:15:09 am -0400	V3.1: 6.1 MEDIUM
CVE-2024-30482 - Cross-Site Request Forgery (CSRF) vulnerability in Brice CAPOBIANCO Simple Revisions Delete.This issue affects Simple Revisions Delete: from n/a through 1.5.3. Published: March 29, 2024; 12:15:09 pm -0400	V3.1: 8.8 HIGH
CVE-2024-31099 - Missing Authorization vulnerability in Averta Shortcodes and extra features for Phlox theme auxin-elements.This issue affects Shortcodes and extra features for Phlox theme: from n/a through 2.15.7.	V3.1: 8.8 HIGH

Source: <https://nvd.nist.gov/>

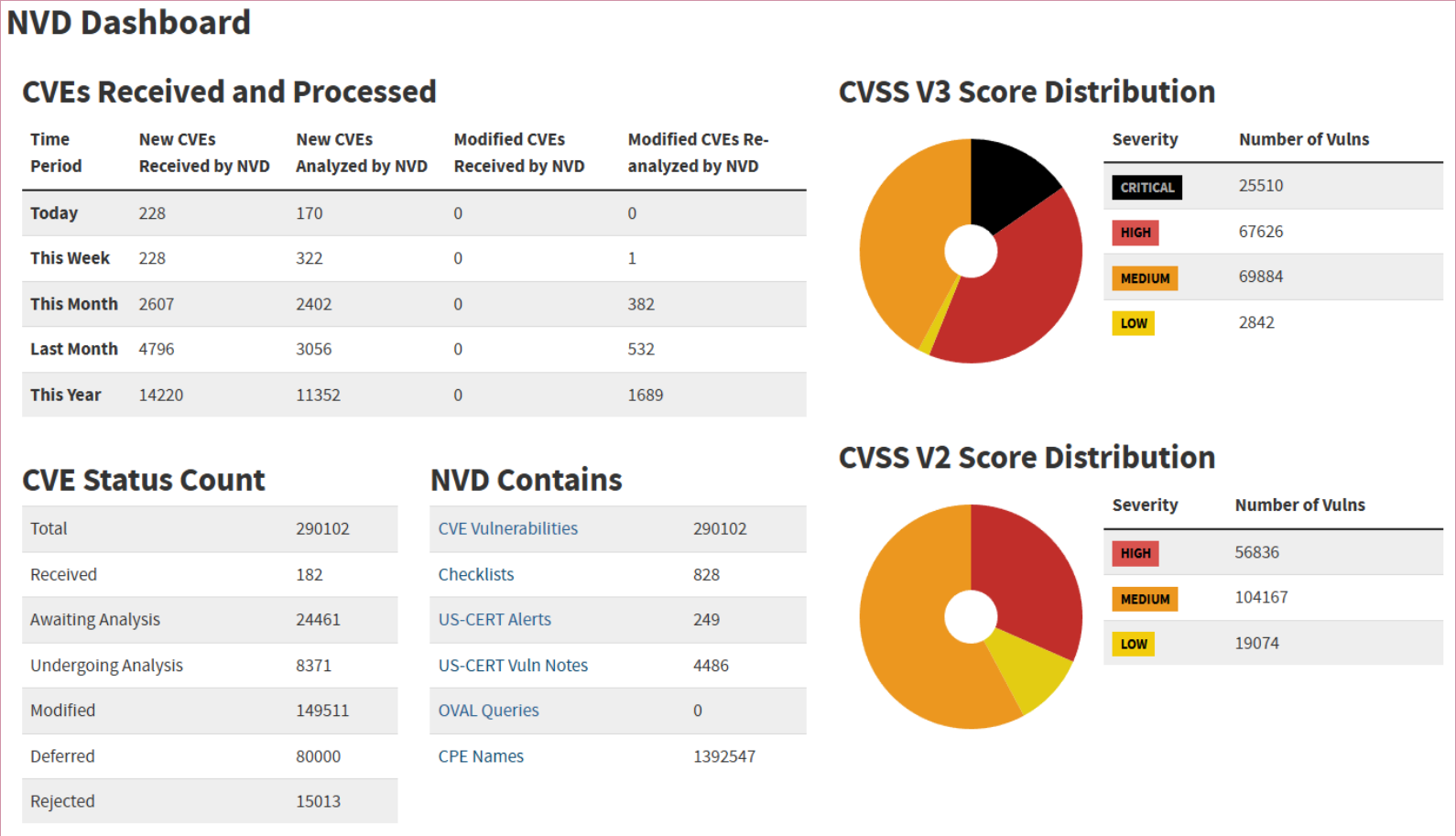
CWE – Common Weakness Enumeration

- A hierarchical list of **software flaw types**.
- Examples:
 - CWE-79: Cross-site scripting (XSS)
 - CWE-89: SQL Injection
- Helps identify root causes and prevent recurring bugs.
- Linked with CVEs in NVD.

1	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') CWE-79 CVEs in KEV: 3 Rank Last Year: 2 (up 1) ▲
2	Out-of-bounds Write CWE-787 CVEs in KEV: 18 Rank Last Year: 1 (down 1) ▼
3	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') CWE-89 CVEs in KEV: 4 Rank Last Year: 3
4	Cross-Site Request Forgery (CSRF) CWE-352 CVEs in KEV: 0 Rank Last Year: 9 (up 5) ▲
5	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') CWE-22 CVEs in KEV: 4 Rank Last Year: 8 (up 3) ▲
6	Out-of-bounds Read CWE-125 CVEs in KEV: 3 Rank Last Year: 7 (up 1) ▲
7	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') CWE-78 CVEs in KEV: 5 Rank Last Year: 5 (down 2) ▼
8	Use After Free CWE-416 CVEs in KEV: 5 Rank Last Year: 4 (down 4) ▼
9	Missing Authorization CWE-862 CVEs in KEV: 0 Rank Last Year: 11 (up 2) ▲
10	Unrestricted Upload of File with Dangerous Type CWE-434 CVEs in KEV: 0 Rank Last Year: 10

NVD – National Vulnerability Database

- A U.S. government repository of vulnerability data.
- Managed by NIST (National Institute of Standards and Technology).



Source: <https://nvd.nist.gov/general/nvd-dashboard>

CVSS – Common Vulnerability Scoring System

✚ What Affects the Score?

- 📶 **Access Vector** – *How does the attack reach the system?*
 - **Network (High risk):** Can attack over the internet (e.g., remote RCE)
 - **Local (Lower risk):** Needs physical or local access (e.g., USB attack)
- 👤 **User Interaction** – *Does it require someone to click or do something?*
 - **None:** Attacker can run it automatically (more dangerous)
 - **Required:** Needs victim to open file or click a link (less severe)
- 🔑 **Privileges Required** – *Does the attacker need to be logged in?*
 - More privileges = lower risk
 - No privileges = higher risk
- ⚡ **Impact** – *What can the attacker do?*
 - Access data, crash system, run code, etc.



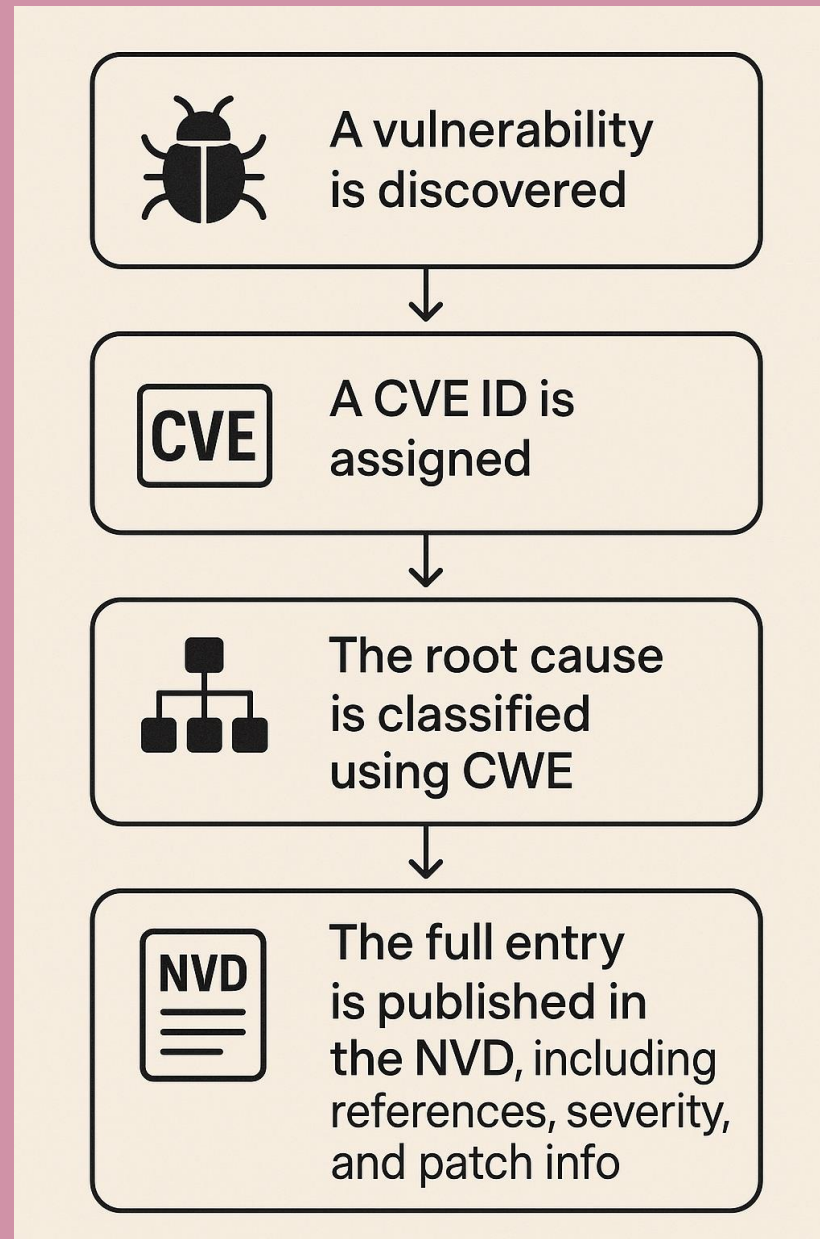
Helps teams prioritize — fix critical/high scores first!



Score Range

- **0.0** – No risk
- **0.1–3.9** – Low
- **4.0–6.9** – Medium
- **7.0–8.9** – High
- **9.0–10.0** – Critical

How they work together?



What can be done using them?

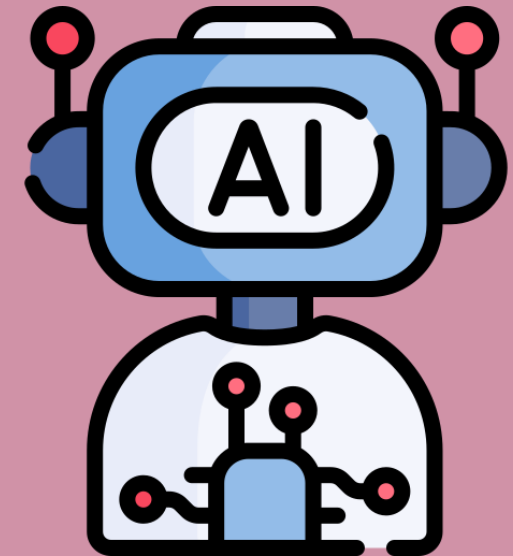
For Developers & Security Teams:

- **TRACK** known vulnerabilities in dependencies (via CVE/NVD)
- **UNDERSTAND** root causes to avoid making the same coding mistakes (CWE)
- **PRIORITIZE** fixes based on CVSS score (e.g., fix CVSS 9+ first)



For Tooling & Automation:

- **Static/dynamic analysis tools** tag issues using CWE
- These data can be used as **labels** to train **machine/deep-learning-based** vulnerability prediction models



COMMON



WEB ATTACKS



USERNAME

PASSWORD

```
SELECT * FROM users
WHERE username = '${username}' AND
      password = '${password}'
```

USERNAME

| \`OR 1=1 --;

PASSWORD

| * * * * *

```
SELECT * FROM users
WHERE username = '${username}' AND
      password = '${password}'
```

```
SELECT * FROM users
WHERE username = '' or 1 = 1; --' AND
      password = '${password}'
```

```
SELECT * FROM users
WHERE username = '${username}' AND
      password = '${password}'
```

```
SELECT * FROM users
WHERE username = '' or 1 = 1 --' AND
      password = '${password}'
```

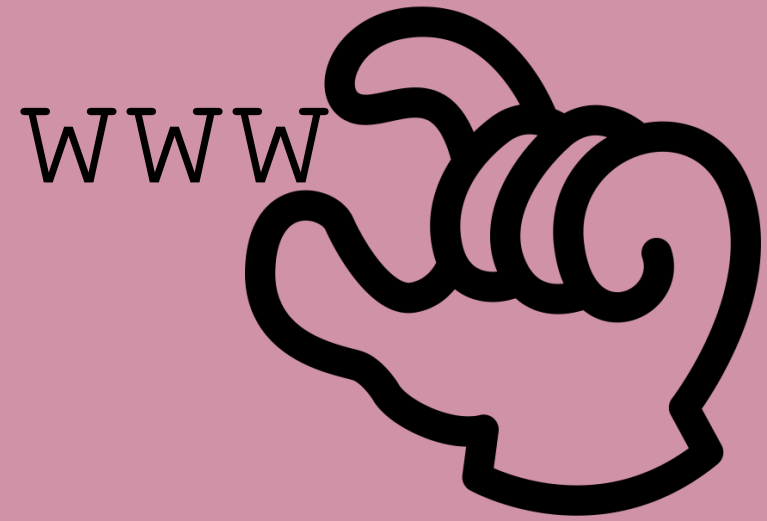


SQL INJ.

Sanitise the input

Use parameterized query

URL



Manipulation/Rewrite

`http://127.0.0.1:5000/profile?user_id=1`

`http://127.0.0.1:5000/profile?user_id=2`

What if we change this?



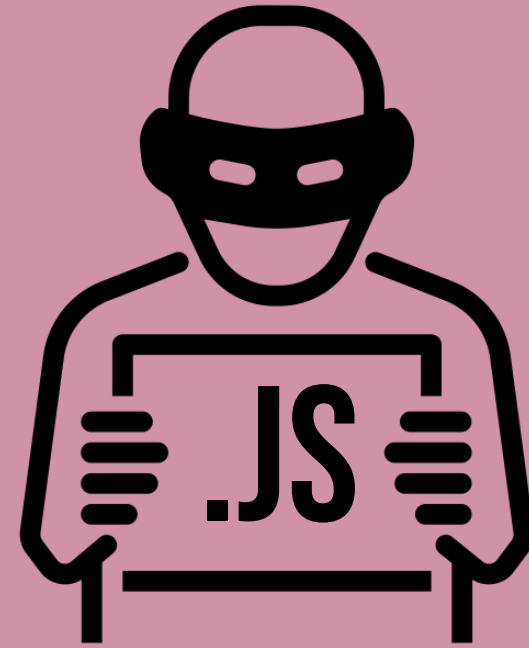
URL MANIPULATION

Never trust URL parameters for access control

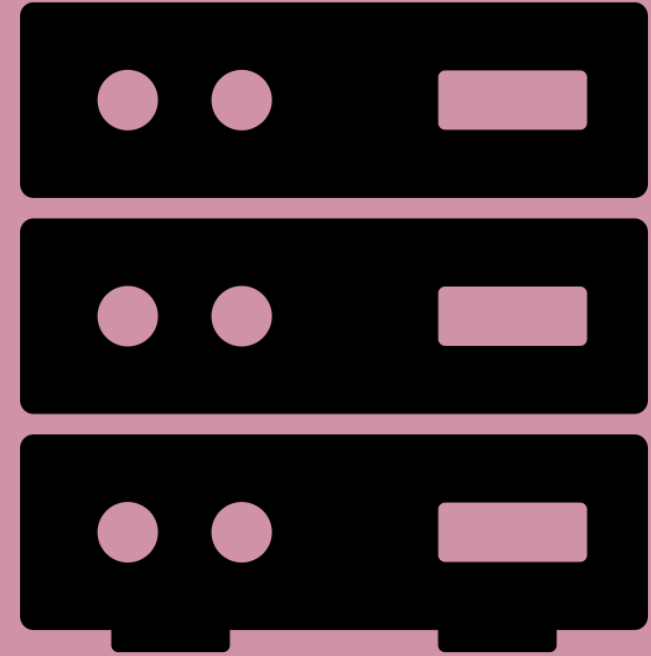
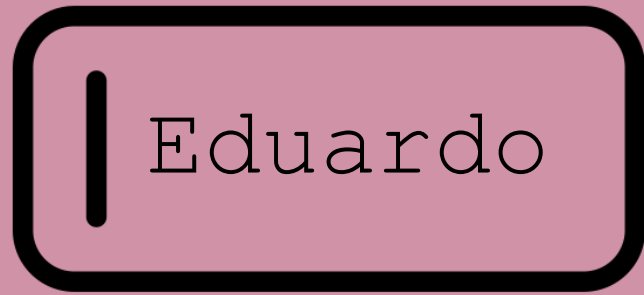
Use authentication checks

Use session-based authentication

XSS



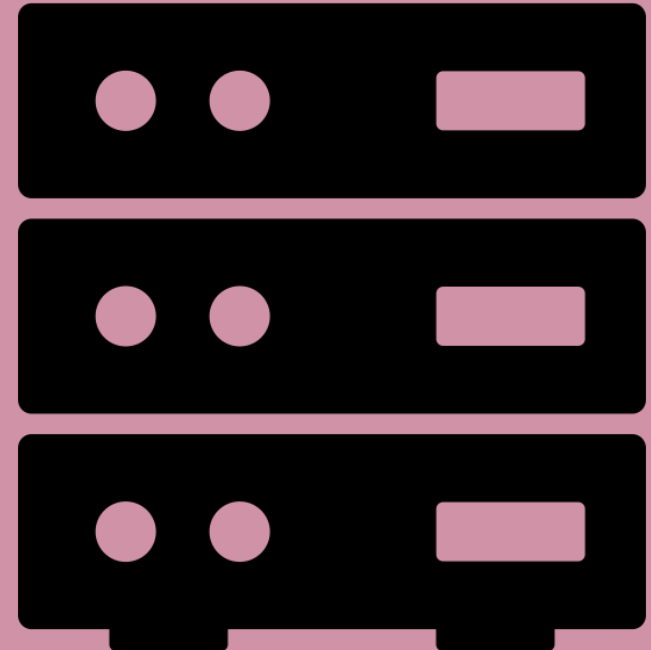
Cross-Site Scripting



Hello, Eduardo



```
<script  
type="text/javascript">  
  alert("You're  
  Hacked!");  
</script>
```



You're Hacked!





XSS

*Sanitise anything that gets
output to the browser*

Look after your POST requests



COOKIE

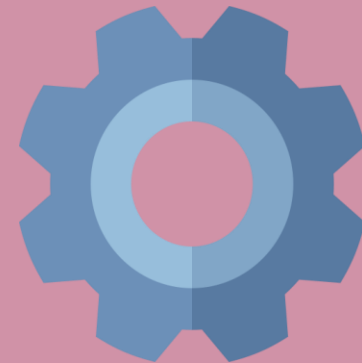
THEFT

What is a Cookie?

- **Small piece of data** stored by the browser, sent with every request to the same site
- Set by the server using Set-Cookie HTTP header
- Helps **track sessions, store preferences, or remember users**

Common Use Cases:

1. Keeping users **logged in**
2. Storing **user settings**
3. Tracking **shopping cart items**



With JavaScript, a cookie can be created like this:

```
document.cookie = "username=John Doe";
```

You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";
```

With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```



Cookie Theft: Comparison of Three Vulnerabilities

Vulnerability	How It Enables Cookie Theft	Key Defense Measures
XSS	Attacker injects JavaScript that accesses document.cookie	- Sanitize user input (html.escape)- Set HttpOnly on cookies
No HTTPS	Cookies sent in plaintext over the network (e.g. public Wi-Fi)	- Use HTTPS (Secure cookie flag)- Redirect all HTTP to HTTPS
Misconfiguration	Cookie lacks proper flags (e.g. missing HttpOnly, Secure, SameSite)	- Set all security flags: HttpOnly, Secure, SameSite- Harden app/server config

What can we do?

DEVOPS + SECURITY

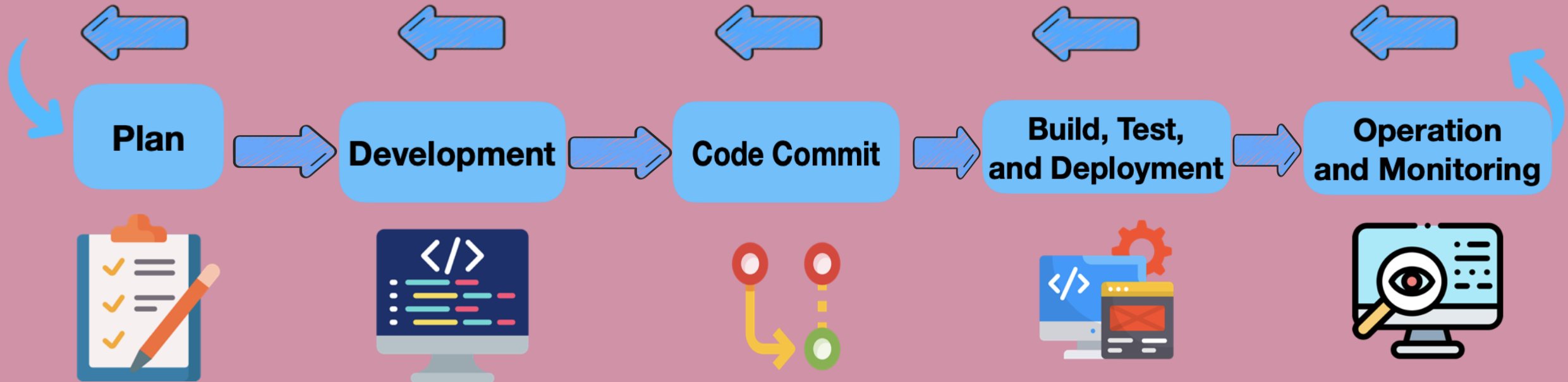
=

DEVSECOPS



DevOps (Development & Operation)

The word “DevOps” was coined in 2009 by Patrick Debois, who became one of its gurus.



- Gather Requirements
- Planning Features
- Task Assignment

Coding and Implementation

Code review and feedback before merging into the main branch.

Transform code to executables and test the App. Then deploy to the server.

Monitor App performance and user feedback.

Plan

Threat Modeling

- **What it is:** A structured way to identify and assess potential security threats to your system.
- **Goal:** Understand *what could go wrong* and how to prevent or mitigate those risks.
- **Common Techniques:**
 - STRIDE (Spoofing, Tampering, Repudiation, etc.)
 - Attack trees or data flow diagrams (DFDs)
- **When to use:** Early in design discussions or feature planning.

Impact Analysis

- **What it is:** Evaluating the potential consequences of a security risk or failure.
- **Goal:** Prioritize threats based on business impact and likelihood.
- **Focus Areas:**
 - Confidentiality, Integrity, Availability (CIA)
 - Regulatory or compliance concerns (e.g., GDPR, HIPAA)
- **Outcome:** Informs which security controls should be built in first.

Development (Static Analysis)

What is Static Analysis?

Static analysis inspects source code *without running it*, to find bugs and security issues early.



Catches issues **before deployment**



Fast and **automated**



Helps enforce secure coding practices



What Can It Find?

- SQL Injection
- Buffer overflows
- Hardcoded secrets
- Insecure API usage
- Many more...

Development (Static Analysis Example)



Our rule for pattern matching

```
def find_sql_injection(lines):  
    findings = []  
    for i, line in enumerate(lines, start=1):  
        if "SELECT" in line and "+" in line and "=" in line:  
            findings.append((i, line.strip(), "Potential SQL Injection"))  
    return findings
```

Case	Description	Flagged?	Correct?
Unsafe	User input concatenated in SQL query	✓ Yes	✓ True Positive
Safe Case (1)	Hardcoded "5" used in SQL query (no input)	✓ Yes	✗ False Positive
Safe Case (2)	Parameterized query with user input	✗ No	✓ Correctly Safe

Development (Static Analysis Example)

Layer	Our Toy Example	Real Analyzer
Line-by-line check	✓ Yes	✓ Yes
Code structure (AST)	✗ No	✓ Yes
Data flow tracking	✗ No	✓ Yes
Input source tracing	✗ No	✓ Yes
Complex logic analysis	✗ No	✓ Yes (sometimes)

- **Code structure (AST)**
 - Parses code into a tree to understand functions, variables, and operations — not just raw text.
- **Data flow tracking**
 - Follows how data moves between variables, functions, and objects — useful for spotting unsafe usage.
- **Input source tracing**
 - Tracks untrusted inputs (e.g., from users or network) and sees where they end up — critical for security.
- **Complex logic analysis**
 - Understands conditions, loops, and different execution paths — helps reduce false positives.

Development

2023 CWE Top 25 Most Dangerous Software Weaknesses

[Top 25 Home](#)[Share via: !\[\]\(99f58673407353e96a019fbca558fd72_img.jpg\)](#)[View in table format](#)[Key Insights](#)[Methodology](#)**1****Out-of-bounds Write****[CWE-787](#)** | CVEs in KEV: 70 | Rank Last Year: 1

ChatGPT Security Demo

Is there any security problem with the code below?

If so, please describe why the code is not secure and suggest a fix.

```
int returnChunkSize(void *) {  
    /* if chunk info is valid, return the size of usable memory,  
    * else, return -1 to indicate an error  
    */  
    ...  
}  
  
int main() {  
    ...  
    memcpy(destBuf, srcBuf, (returnChunkSize(destBuf) - 1));  
    ...  
}
```

WHY DANGEROUS?

- **Memory Corruption**
- **Easy to Exploit**
- **Commonly Exploited**
- **Hard to Detect**
- **Wide Impact in the System**

Code Commit



Security Copilot – Code Scanning Autofix

- Generate **fix code** suggestion with **natural language explanation** to automate PR workflows



Build, Test, Deployment

Configuration Validation & Infrastructure Scanning

- **What it is:** Ensuring that both configurations and infrastructure are secure, properly set up, and free from vulnerabilities.
- **Goal:** Detect misconfigurations and vulnerabilities in both code and live environments.
- **Common Tools:**
 - **Checkov** / **Kube-score** (for cloud and Kubernetes config validation)
 - **Aqua Security** / **Trivy** (for container scanning)
- **Actions:**
 - Validate security settings for **cloud resources**, **network rules**, **permissions**.
 - Scan **Infrastructure as Code (IaC)** for vulnerabilities.
 - Check **containers** for security weaknesses and outdated dependencies.
- **Outcome:** Proactively detect and fix misconfigurations and vulnerabilities before deployment.

Operation and Monitoring



DataDog: Cloud Monitoring as a Service

<https://www.datadoghq.com/>

- Real-time App performance monitoring, threat detection, and anomaly detection
- Integrated dashboard to observe infrastructures, system logs, and service dependencies



HANDS-ON ACTIVITIES



The “Fishing Pole”

A Unified Security Risk Evaluation Framework

Why Every Team Needs a Custom Security Plan?

- Different apps » different risks
- Web app with user login ≠ Chatbot answering FAQs
- One-size-fits-all security doesn't work

◆ Core Idea:

A **simple, adaptable framework** that you can fill out based on your specific project (web app, mobile app, chatbot, etc.).

Goals of the Framework:

- Think like a **security-conscious** developer
- Take **ownership of your app's threat landscape**
- Learn to **document security** considerations

The Risk Evaluation Framework







("The Fishing Pole")

RISK / THREAT	TRIGGER / CAUSE	LIKELIHOOD	IMPACT	CONTINGENCY PLAN
e.g., SQL injection	User input is passed directly to SQL query	Medium	High (data breach, DB corruption)	Use input validation, prepared statements
e.g., Hardcoded API key	Developer pushes code with secrets to GitHub	Low	High (unauthorized access)	Use env variables, secret scanners
e.g., No HTTPS	API endpoint uses HTTP	High	Medium (data in transit exposed)	Enforce HTTPS via server config
e.g., Lack of authentication	Public-facing admin route	Medium	High (unauthorized access)	Add auth middleware, access control

Explanation of Columns

- **RISK / THREAT:** A specific vulnerability or problem (e.g., insecure login, exposed data).
- **TRIGGER / CAUSE:** What could make this risk occur? (e.g., bad coding practice, misconfig, lack of validation).
- **LIKELIHOOD:** How likely is it to happen? (Low / Medium / High).
- **IMPACT:** What damage could it cause if it happens? (Low / Medium / High).
- **CONTINGENCY PLAN:** What will your team do to prevent, detect, or respond to the issue?

Hint 1: Common Risk Categories Checklist

- ☐  **Authentication & Authorization** (e.g., weak login, privilege escalation)
- ☐  **Input Validation** (e.g., injection attacks)
- ☐  **Secrets Management** (e.g., hardcoded keys, exposed tokens)
- ☐  **Data in Transit** (e.g., unencrypted API calls)
- ☐  **Storage Security** (e.g., insecure databases, open S3 buckets)
- ☐  **3rd Party Components** (e.g., outdated libraries, vulnerable dependencies)

 **Ask yourself:**

“Do any of these apply to our app?”

“Where in our design or code could these go wrong?”

Hint 2 (High Level): Review the Design of the Software



Look at your **system diagram or flowchart** and ask:

- What are the **components**? (e.g., frontend, backend, APIs, DB)
- What are the **data flows**?
- Where are the **trust boundaries**? (e.g., user-to-server, server-to-DB)

Use your existing UML, sequence diagrams, or data flow diagrams



Look for entry points, sensitive data paths, and external interfaces

Hint 3 (Low Level): Review Specific Code or Functions



Zoom in...

- What are the **critical features** (login, file upload, form submission)?
- Which **functions or endpoints** handle user input or access data?