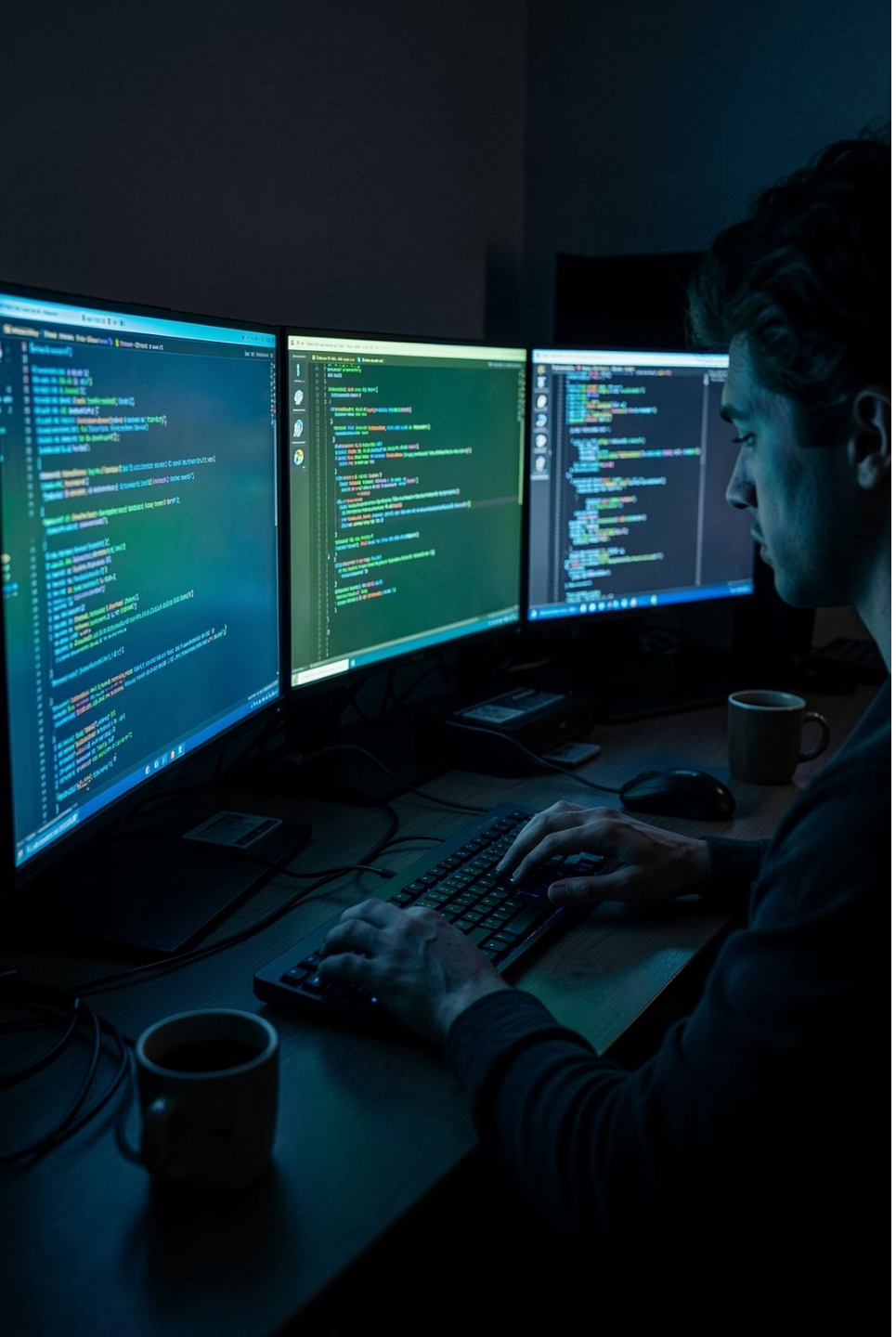
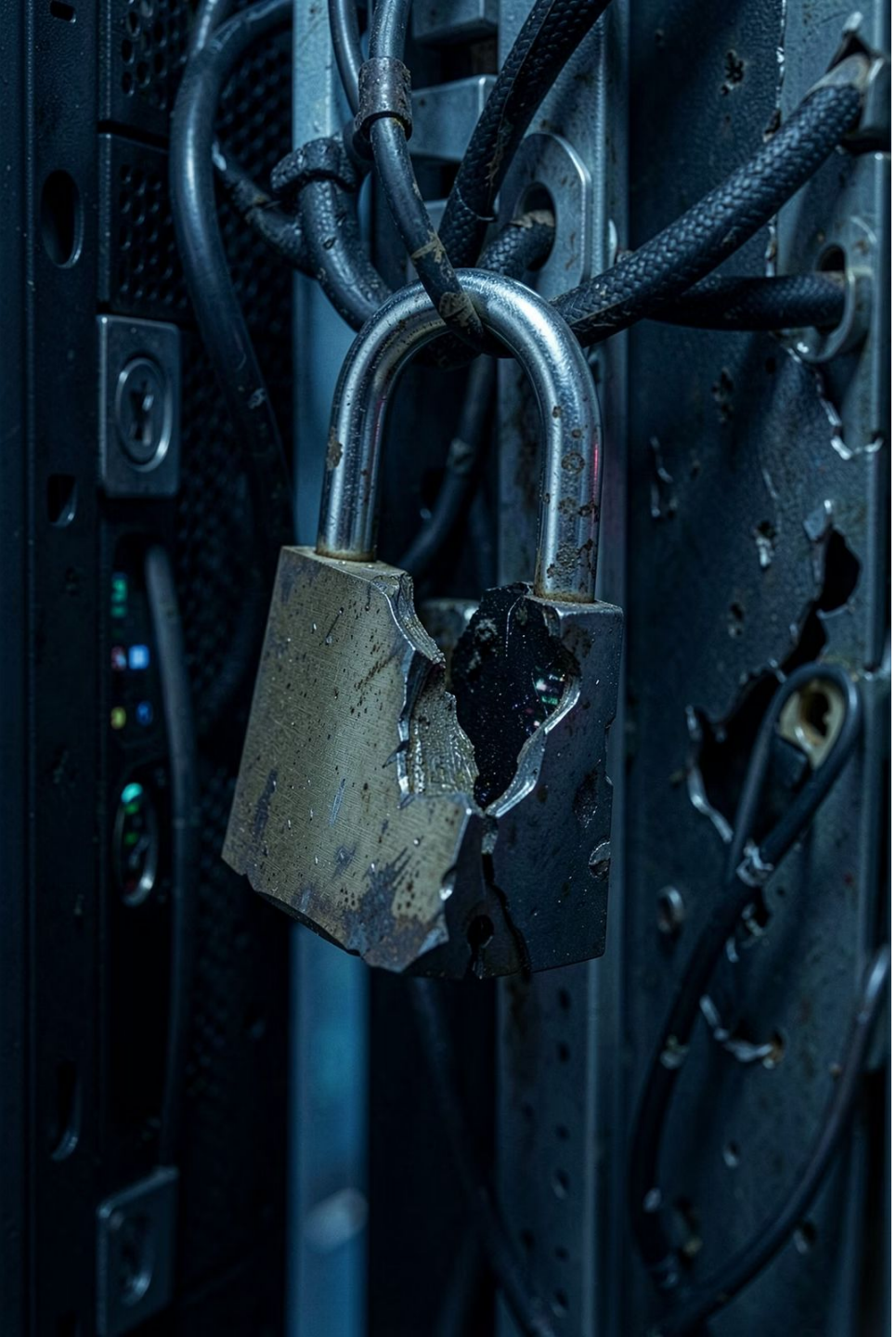


# Secure Software Development Lifecycle (SSDLC) with Hands-On Coding

**Cyber School - Professional Training for Adults**

A 4.5-hour hands-on workshop: build security into every line of code.

CYBER SCHOOL™



# Why Security Cannot Wait

## Bolted-On Security Fails

Security bolted on at the end of a project is expensive, fragile, and frequently too late. Fixing a vulnerability in production costs far more than preventing it at design time.

## Real-World Proof

The MOVEit file-transfer breach and the T-Mobile data breach exposed hundreds of millions of records - consequences of security treated as an afterthought.

## What You Will Learn Today

Build security in - phase by phase, with working code. Every session delivers practical skills you can apply immediately.

CYBER SCHOOL™

# Workshop Agenda

### Session 1: Introduction to SSDLC

30 min

### Session 2: Phase-by-Phase Breakdown with Code

60 min

### Session 3: Hands-On Threat Modeling

45 min

### Session 4: Secure Coding Practices Workshop

45 min

### Session 5: Security Testing Hands-On

45 min

### Session 6: Integrating DevSecOps and Wrap-Up

45 min

CYBER SCHOOL™

# What is the SDLC?

The Software Development Lifecycle: a well-defined process to design, develop, test, and deploy software.

1. **Requirements**
2. **Design**
3. **Development**
4. **Testing**
5. **Deployment**
6. **Maintenance**

⚠ Traditionally, security enters only at the end - if at all. That is the problem we are solving today.

CYBER SCHOOL™

# From SDLC to SSDLC

SSDLC means security is a fundamental consideration in every phase, not an afterthought.

### Requirements

Define security requirements up front.

### Design

Model threats before writing code.

### Development

Secure coding standards applied consistently.

### Testing

Automated security testing at every build.

### Deployment

Hardening and monitoring from day one.

### Maintenance

Continuous patching throughout the product lifetime.

# Three Principles Behind Every Decision

## CIA Triad

**Confidentiality, Integrity, Availability** - the foundation of information security. Every security decision maps back to one or more of these three properties.

## Principle of Least Privilege

Grant only the minimum access required. When a component is compromised, damage stays contained to the smallest possible blast radius.

## Defense in Depth

Multiple overlapping layers of security controls. When one layer fails, others hold. No single point of failure.

CYBER SCHOOL™

# Case Studies: When the Lifecycle Fails

## MOVEit File-Transfer Breach

A single vulnerability in a widely-used file-transfer application resulted in hundreds of organizations being compromised. Sensitive data from government agencies, financial institutions, and healthcare providers was exposed worldwide.

## T-Mobile Data Breach

Millions of customer records were exposed - names, Social Security numbers, driver's license data, and more. A failure to secure internal systems allowed attackers to move laterally and exfiltrate at scale.

Group activity: for each breach - in which SDLC phase was the vulnerability introduced? Discuss with your table and be ready to present your analysis.

CYBER SCHOOL™

# Phase 1: Requirements Gathering

Security requirements live next to functional requirements. Use cases describe how the system should work; misuse cases describe how an attacker abuses it.

## Example - Authentication Requirements

### Multi-Factor Authentication

MFA required for all users - no exceptions. A password alone is not sufficient for any account with access to sensitive data.

### Password Hashing

bcrypt password hashing with cost factor 12. Never store plaintext or weakly hashed passwords.

### Account Lockout

Lockout after 5 failed attempts. Prevents brute-force and credential-stuffing attacks at the authentication layer.

# Phase 2: Secure Design and Threat Modeling

## Threat Modeling in Four Steps

### Identify Assets

What data and systems must be protected?

### Identify Threats

Who are the adversaries and what are their goals?

### Analyze Risk and Impact

Likelihood multiplied by impact equals priority.

### Design Mitigations

Controls that reduce risk to an acceptable level.

## Design Principles

- Separation of duties
- Fail-secure defaults
- Defense in depth

## Tools

- Microsoft Threat Modeling Tool
- OWASP Threat Dragon

## Example: File Upload

Restrict file types, verify magic-byte headers, scan for malware before storing or processing any uploaded content.

CYBER SCHOOL™

# Phase 3: Secure Development - SQL Injection

SQL injection remains one of the most exploited vulnerability classes. The fix is straightforward - but only if you know what to look for.

## Insecure Code

```
cursor.execute(
  "SELECT * FROM users "
  "WHERE username = '"
  + username + "'"
)
```

An attacker submits `' OR '1'='1` and walks straight in. The database cannot distinguish attacker input from legitimate SQL.

## Secure Code

```
cursor.execute(
  "SELECT * FROM users "
  "WHERE username = %s",
  (username,)
)
```

Parameterized queries separate code from data. The database driver handles escaping. Always use parameterized queries - no exceptions.

Parameterized queries are not optional. They are the baseline. Every database interaction must use them.

CYBER SCHOOL™

# Phase 3: Secure Development - XSS and Output Encoding

## Insecure Code

```
echo $_GET['name'];
```

The browser executes whatever arrives in the query string. An attacker injects a script tag and the victim's browser runs malicious JavaScript.

## Secure Code

```
echo htmlspecialchars(
  $_GET['name'],
  ENT_QUOTES,
  'UTF-8'
);
```

Input becomes harmless text. The browser renders it as a string, not as executable code.

## Rules of Thumb

### Validate All Input

Reject anything that does not conform to the expected format, length, and character set.

### Encode All Output

Context-aware encoding for HTML, JavaScript, CSS, and URL contexts.

### Handle Errors Safely

Never leak stack traces, database errors, or internal paths to the client.

## Standards

OWASP Top 10 and CERT Secure Coding do the heavy lifting - know them and apply them.

CYBER SCHOOL™

# Phase 4: Security Testing

### SAST

Static analysis: scans source code without running it. Catches vulnerabilities early in the development cycle. Tool: **SonarQube**.

### DAST

Dynamic analysis: attacks the running application from the outside, simulating a real attacker. Tool: **OWASP ZAP**.

### IAST

Interactive analysis: observes from inside the application while it runs, combining the strengths of SAST and DAST.

### Penetration Testing

Simulates a real attacker before a real attacker arrives. Validates that controls work under adversarial conditions.

CYBER SCHOOL™

# Phase 5: Deployment and Monitoring

## Environment Hardening

- Remove unnecessary services and packages
- Least privilege for all deploy credentials
- Close unused ports
- Patch continuously - do not wait for scheduled windows
- Keep secrets out of the codebase entirely

## Monitoring Stack Options

- Graylog
- Splunk
- ELK Stack (Elasticsearch, Logstash, Kibana)
- Datadog

## Example: Logging Failed Login Attempts

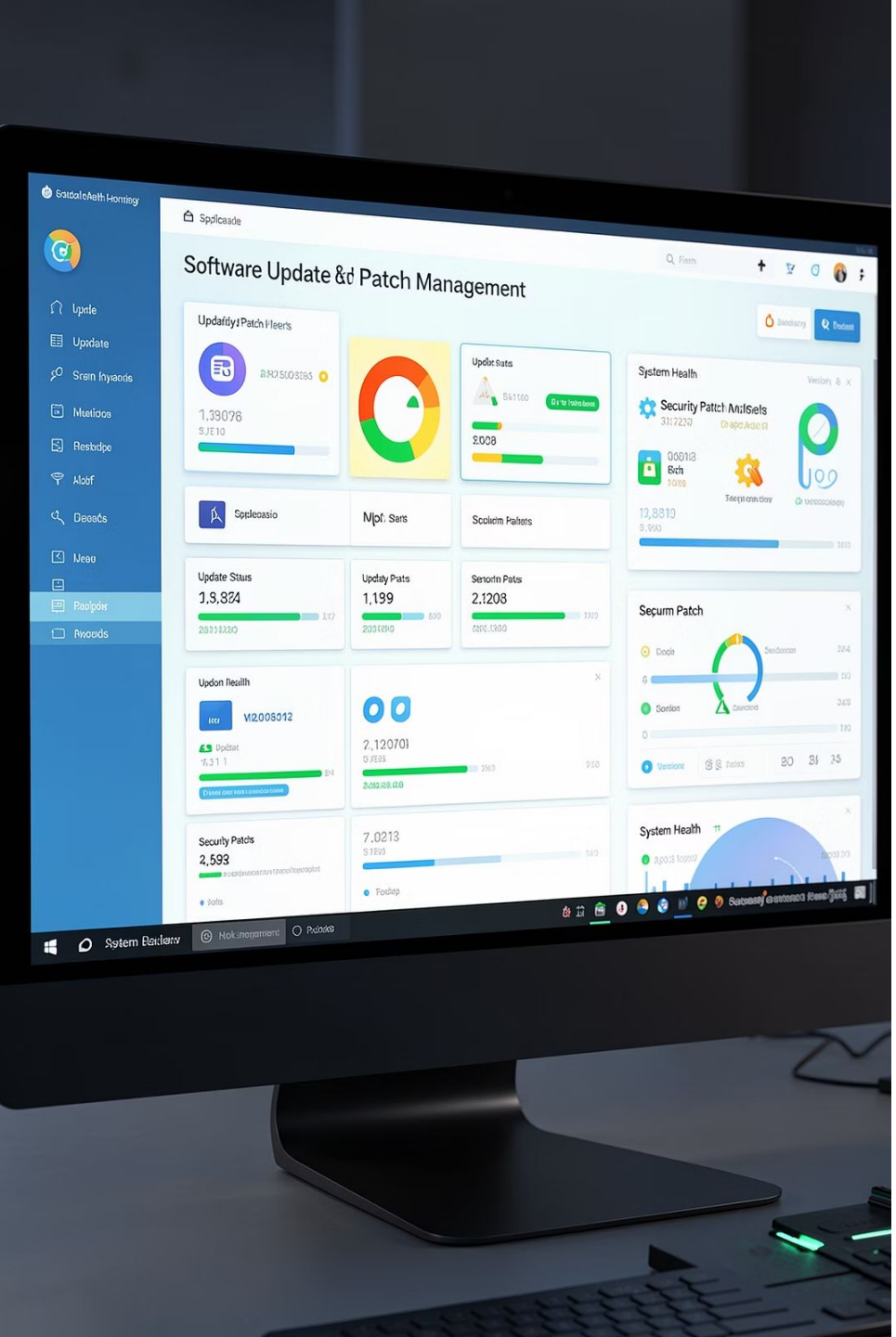
```
import logging

logger = logging.getLogger(__name__)

def login(username, success):
    if not success:
        logger.warning(
            "Failed login attempt "
            "for user: %s",
            username
        )
```

Three lines of Python. Every failed attempt is recorded with a timestamp, username, and source IP - ready for your SIEM to alert on.

CYBER SCHOOL™




# Phase 6: Maintenance

Security does not end at launch. The maintenance phase is where most organizations fall behind - and where attackers find their opportunities.

### Continuous Patching

Apply security patches promptly. Unpatched known vulnerabilities are the most common attack vector in enterprise breaches.

### Dependency Updates

Third-party libraries carry their own vulnerabilities. Your application inherits every CVE in every dependency you ship.

### Vulnerability Management

Track, prioritize, and remediate vulnerabilities systematically. A backlog of unaddressed findings is a liability.

### Dependabot

Automated detection and fixes for vulnerable libraries. Integrates directly into GitHub workflows and opens pull requests automatically.

The cheapest incident is the one your update pipeline prevented.

CYBER SCHOOL™

# Hands-On 1: Threat Modeling

45 MINUTES

## Scenario

You receive the system diagram of an e-commerce platform - user authentication, product catalog, payment processing, order management, and third-party integrations.

## Deliverable

Present your completed threat model and proposed mitigations to the group. Be prepared to defend your risk prioritization decisions.

## Your Task

### 1 Identify Threats

Data exposure, weak authentication, injection points, insecure third-party integrations.

### 2 Build the Threat Model

Use Microsoft Threat Modeling Tool or OWASP Threat Dragon to document your findings.

### 3 Design Mitigations

For each identified threat, propose a concrete control that reduces the risk.

CYBER SCHOOL™

# Hands-On 2: Secure Coding Challenge

45 MINUTES

You receive insecure code snippets covering four vulnerability classes. Find every flaw and fix it.

### SQL Injection

Identify the unsanitized query and replace it with a parameterized equivalent.

### XSS

Find the unencoded output and apply context-aware encoding.

### Buffer Overflow (C)

Replace unsafe string operations with `strncpy` using explicit bounds and null termination.

### Missing CSRF Protection

Implement token-based form protection to prevent cross-site request forgery.

## Buffer Overflow Fix

```
// Unsafe
strcpy(dest, src);

// Safe
strncpy(dest, src, sizeof(dest) - 1);
dest[sizeof(dest) - 1] = '\0';
```

## CSRF Fix

```
<form method="POST">
  <input type="hidden"
    name="csrf_token"
    value="{{ csrf_token }}">
  ...
</form>
```

CYBER SCHOOL™

# Hands-On 3: Security Testing Lab

45 MINUTES

## Target Application

A deliberately vulnerable web application: **OWASP Juice Shop**. It contains intentional vulnerabilities across every OWASP Top 10 category - a safe, legal environment to practice real attack and defense techniques.

## Your Tools

- OWASP ZAP - automated scanning and manual interception proxy
- Burp Suite - advanced web application security testing

## Vulnerabilities to Find

- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- SQL Injection

## CI/CD Integration Question

After completing your manual testing, answer this critical design question:

**Where do these automated scans belong in your CI/CD pipeline?**

- On every commit to a feature branch?
- On every pull request before merge?
- On every deployment to staging?
- All of the above?

Be ready to justify your answer with the tradeoffs between speed and coverage.

CYBER SCHOOL™

# Integrating DevSecOps

DevSecOps: security inside the CI/CD pipeline, running on every commit. Security is no longer a gate at the end - it is a continuous automated process.

### 1 Commit

Developer pushes code. SAST scan runs automatically via SonarQube or Semgrep.

### 2 Build

Dependency scanning checks all third-party libraries for known CVEs.

### 3 Test

DAST scan via OWASP ZAP attacks the running application in a staging environment.

### 4 Deploy

Automated in Jenkins or Azure Pipelines. Only clean builds reach production.

### 5 Monitor

Real-time monitoring with ELK Stack or Datadog closes the feedback loop.

Design exercise: sketch the DevSecOps pipeline for your own organization. What tools do you already have? What gaps exist?

# Key Takeaways

### Security in Every Phase

Security must be integrated into every phase of the SDLC - from requirements gathering through maintenance. It is not a final step.

### Secure Coding Prevents Vulnerabilities

Secure coding practices prevent most vulnerabilities before they exist. Parameterized queries, output encoding, and input validation are non-negotiable baselines.

### Automate and Monitor

Automated testing and continuous monitoring keep applications secure over time. Manual review alone cannot scale to the pace of modern development.

---

**Questions and discussion - then your knowledge check quiz.**

Thank you for attending. Apply what you have learned starting with your next commit.

CYBER SCHOOL™