CST 2412 Module: Programs, Web, and Operating System Security

- Textbook focus: Chapter 3, Chapter 4, Chapter 5
- Theme: Data security, privacy, and ethics in real systems
- Includes: quick lab, applied scenario analysis, and career pathway mapping

How This Builds on Weeks 1 and 2

- Week 1: CIA triad, threats, vulnerabilities, legal/ethical framing
- Week 2: risk management, security toolbox, secure programming bridge
- This module: apply CIA + risk + controls to software, web, and OS layers

Agenda

- Part 1: Chapter 3 - Programs and programming security
- Part 2: Chapter 4 - Web and email attacks on users
- Part 3: Chapter 5 - Operating system security and rootkits
- Part 4: Applied scenario walkthrough
- Part 5: Quick lab and career-connected wrap-up

Learning Objectives

- Differentiate programming flaws from malicious code and explain CIA impacts
- Analyze browser, web, and email attack paths and propose mitigations
- Explain operating system security architecture, reference monitor, and rootkits
- Apply risk, ethics, and governance analysis to a real breach
- Connect course topics to entry-level cybersecurity roles and credentials

Essential Questions

- When does a normal software bug become a security vulnerability?
- Why do user-side web attacks keep succeeding even with better tools?
- What makes operating system compromise uniquely dangerous?
- How do technical decisions become ethical decisions in practice?

Textbook Roadmap for This Module

- Chapter 3: Programs and Programming (textbook pages 160-259)
- Chapter 4: The Web - User Side (textbook pages 260-306)
- Chapter 5: Operating Systems (textbook pages 307-366)
- Class emphasis: defensive reasoning, ethics, and workforce readiness

Chapter 3 Overview: Programs and Programming

- Unintentional flaws: overflow, mediation failures, race conditions
- Malicious code: viruses, worms, Trojan horses
- Countermeasures for users and developers
- Bottom line: software quality is a security issue

Error, Fault, Failure: Why Terminology Matters

- Error: human mistake during development
- Fault: defect in code/design/documentation
- Failure: observed deviation from intended behavior
- Security teams often use the umbrella term "flaw"

Programming Flaw Spotlight: Buffer Overflow

- Data exceeds allocated memory boundaries
- Can cause crash, corruption, or control-flow hijack
- Historically common in low-level memory-unsafe code
- Defense: bounds checks, safer APIs, memory-safe language choices

Programming Flaw Spotlight: Incomplete Mediation

- Server trusts sensitive values controlled by client input
- Example pattern: client-supplied price or total in URL parameters
- Result: integrity failures, fraud, and undetected long-term loss
- Defense: server-side recomputation and full policy checks

Programming Flaw Spotlight: TOCTTOU

- Time-of-check to time-of-use race condition
- State changes after validation but before action
- Produces access control failure despite apparent checks
- Defense: atomic operations, ownership of validated data, locking

Other High-Frequency Flaws in Chapter 3

- Undocumented access points (backdoors or trapdoors)
- Off-by-one indexing and boundary errors
- Integer overflow or wraparound logic failures
- Unterminated strings and parsing confusion

Malicious Code in Chapter 3

- Malware includes viruses, worms, Trojan horses, and variants
- Focus on mechanism and impact, not only naming
- Propagation, persistence, concealment, and payload matter most
- Benign-looking software can still include malicious behavior

Virus vs Worm vs Trojan Horse

- Virus: infects other files/programs and replicates
- Worm: self-propagates across networks
- Trojan horse: hidden malicious function in benign-looking software
- Real incidents often combine multiple behaviors

Malware Economics: Zero-Day and Toolkits

- Many attacks use known, unpatched vulnerabilities
- Zero-day: exploit before a fix is available or deployed
- Toolkits lower attacker skill barriers (script-kiddie effect)
- Defense challenge: defenders must cover many paths, attacker needs one

Countermeasures for Users (Chapter 3)

- Practice software hygiene and source skepticism
- Patch promptly with change control awareness
- Use backups and recoverable system images
- Understand limits of antivirus and signature-based detection

Countermeasures for Developers (Chapter 3)

- Modularity and encapsulation reduce blast radius
- Input validation at trust boundaries
- Complete mediation for every sensitive action
- Secure code review, testing, and threat-informed design

Countermeasures That Commonly Fail

- Security by obscurity (hidden backdoors, hidden assumptions)
- Patch-and-pray without asset visibility
- Checklist compliance without threat modeling
- Over-reliance on one tool category

Chapter 3 Checkpoint

- Prompt 1: Name one flaw and map it to CIA impact
- Prompt 2: Describe one user control and one developer control
- Prompt 3: Explain why malware naming is less useful than behavior analysis

Chapter 4 Overview: The Web - User Side

- Browser attack surface
- Web content attacks targeting users
- Data extraction attacks through script/injection
- Email attack vectors and trust abuse

Why Browsers Are High-Risk by Design

- Browsers fetch from many unseen origins
- Rich scripts execute with significant local privilege
- Users cannot easily inspect full traffic behavior
- Add-ons and plug-ins expand attack surface

Man-in-the-Browser Attacks

- Malicious code operates inside trusted browser context
- Can intercept credentials before encryption
- Can alter transaction details and displayed confirmations
- Traditional page-level trust signals may still appear normal

Page-in-the-Middle and Download Substitution

- Victim is redirected to attacker-controlled page
- Fake login or update pages capture credentials
- User-approved install can still deliver malicious payload
- Trusting UI appearance is insufficient

Defacement, Fake Sites, and Fake Software

- Website defacement can be obvious or subtle
- Fake domains and cloned assets enable convincing impersonation
- Compromised trusted sites amplify attacker success
- Integrity verification and signed distribution help

Web Bugs and Cross-Site Tracking

- Tiny invisible resources can trigger tracking events
- Third-party trackers correlate behavior across sites
- Metadata leakage creates privacy risk beyond content theft
- Security and privacy concerns overlap directly

Cross-Site Scripting (XSS)

- Untrusted input is executed as script
- Reflected and stored/persistent variants
- Can steal tokens, alter page behavior, and pivot attacks
- Defenses: output encoding, input handling, CSP, framework defaults

SQL Injection

- Attackers inject SQL through unsanitized input
- Can expose or modify unauthorized records
- Often caused by string concatenation in query building
- Defenses: parameterized queries and least-privilege DB accounts

Path Traversal and Server-Side Include Abuse

- Dot-dot-slash patterns can escape intended file directories
- Server-side include commands may execute unsafe actions
- Poor isolation magnifies impact of compromise
- Defenses: allowlists, jailed runtime, constrained execution context

Website Breaches Are Also User Harm

- Credential stores are high-value attacker targets
- Password reuse multiplies downstream compromise
- Users suffer even when site owner was initially breached
- Defenses: strong unique passwords, MFA, breach response discipline

Email Attacks: Spam, Spoofing, Phishing, Spear Phishing

- High volume and low cost make spam economically persistent
- Header fields are easy to spoof without strong validation
- Phishing solicits sensitive data or unsafe actions
- Spear phishing personalizes bait using social context

Defensive Strategy for Chapter 4 Threats

- Secure coding: validate, encode, parameterize, constrain
- Architecture: segmentation, least privilege, monitoring
- User controls: skepticism, verification, reporting
- Cryptographic protections for email authenticity/confidentiality

Chapter 4 Checkpoint

- Prompt 1: Compare MITB and page-in-the-middle
- Prompt 2: Explain one injection attack and one defense
- Prompt 3: Why can phishing succeed even with technical controls?

Chapter 5 Overview: Operating Systems

- OS enforces separation, access control, and resource governance
- OS security quality affects all upper-layer applications
- Trusted design principles reduce systemic risk
- Rootkits show what happens when OS trust is subverted

OS as First Line of Defense and Prime Target

- Authentication, memory isolation, file controls, process control
- Boot sequence timing affects who controls system first
- Early compromise can neutralize later defenses
- Privilege escalation aims for administrator or root authority

Security Features in Ordinary Operating Systems

- User authentication and credential handling
- Memory protection via paging/segmentation mechanisms
- File, device, and object access control
- Auditing and accountability records

Memory Protection and Isolation

- Base/bounds concepts and protected address spaces
- Execute vs read/write permissions for memory regions
- Isolation limits one process from corrupting another
- Design goal: containment of faults and attacks

Virtualization and Hypervisors

- Virtualization presents controlled resource views per workload
- Hypervisor mediates guest access to physical resources
- Supports isolation, legacy compatibility, and safer testing
- Security benefit depends on strong boundary enforcement

Sandbox and Honeypot Concepts

- Sandbox: constrained execution space for untrusted code
- Honeypot: monitored decoy environment to study attacker behavior
- Both rely on strong isolation from production assets
- Useful for education, detection, and threat intelligence

Layered Design in Secure Operating Systems

- Critical functions placed in lower, more trusted layers
- Higher layers consume lower-layer services
- Layering supports modularity, analysis, and damage containment
- Poor layering increases hidden coupling and vulnerability

Security Kernel and Reference Monitor

- Security kernel: locus of security enforcement
- Reference monitor properties: tamperproof, unbypassable, analyzable
- Every protected access should pass through mediation
- Small, testable enforcement core improves assurance

Correctness, Completeness, and Secure Design

- Correctness: controls behave as intended
- Completeness: controls exist wherever needed
- Saltzer and Schroeder principles remain relevant
- Security must be built in from design, not bolted on later

Trusted Systems and Evaluation

- "Trusted" means justified confidence, not blind belief
- Trusted computing base (TCB) must be minimal and defensible
- Independent evaluation frameworks improve assurance
- Evidence-based trust supports high-stakes environments

Rootkits: Core Idea

- Rootkits seek highest privilege and long-term persistence
- Operate by modifying or intercepting OS behavior
- Goal: hide malicious activity from normal tools
- Can neutralize logging, scanning, and startup controls

Rootkit Stealth Techniques

- Hooking API/system-call paths to filter outputs
- Hiding files, processes, registry keys, or network activity
- Splicing malicious control flow into trusted routines
- Loading early in boot chain for control precedence

Rootkit Examples from Chapter 5

- Phone rootkit demonstrations (microphone, GPS, battery abuse)
- Sony XCP controversy: hidden DRM behavior and secondary risk
- TDSS family: evolving stealth and command-and-control techniques
- Lesson: complexity and opacity increase systemic fragility

Rootkit Defense Playbook

- Secure boot and code-signing enforcement
- Patch management and attack-surface reduction
- Least privilege and hardened endpoint configuration
- Out-of-band integrity checks and incident recovery plans

Chapter 5 Checkpoint

- Prompt 1: Why is reference monitor quality central to OS security?
- Prompt 2: Give one rootkit stealth mechanism and one defense
- Prompt 3: Explain why trusted means evidence-based

Applied Scenario Setup: Student Support Portal

- Fictional but realistic campus data system
- Stores profiles, appointment notes, and referral metadata
- Directly maps to Chapters 3, 4, and early 5
- Goal: diagnose risk and prioritize practical controls

Incident Timeline (Hypothetical Academic Term)

- Week 1: new feature release under deadline pressure
- Week 3: anomalous login and query activity appears
- Week 4: unauthorized export behavior is detected
- Week 5: containment, communication, and remediation actions begin

Trust Boundaries and Data Flow

- Browser and mobile clients send requests to web application
- Application calls auth service, database, and file storage
- Admin portal and API endpoints share core backend resources
- Each boundary requires validation, authorization, and auditing

Vulnerability Profile: What Went Wrong

- Incomplete mediation on one API route
- Unsafe query construction in a legacy code path
- Over-privileged service account increased blast radius
- Logging existed but alerting thresholds were weak

Attack Path and Impact Progression

- Initial foothold through unvalidated input path
- Privilege abuse via broad backend permissions
- Sensitive record enumeration and staged export attempts
- Primary impacts were confidentiality and integrity

Detection and Response Gaps

- Alert fatigue delayed escalation decisions
- Asset and dependency inventory was incomplete
- Incident roles and handoffs were unclear
- Communication plan was reactive, not rehearsed

Privacy and Ethics Analysis

- Sensitive student data creates asymmetric harm risk
- Duty of care includes prevention and transparent response
- Notification quality affects user recovery and trust
- Ethical handling requires support, not only technical fixes

Control Prioritization: First 30 Days

- Week 1: fix mediation gaps and enforce parameterization
- Week 2: reduce service privileges and segment critical paths
- Week 3: tune detections and rehearse escalation workflow
- Week 4: complete policy updates and post-incident review

Applied Scenario Debrief: Reusable Lessons

- Secure coding removes entry paths before deployment
- Operations and governance determine incident scale
- Detection must be actionable, not just voluminous
- Ethics and communication are part of security quality

Quick Lab: Overview

- Duration: 20-25 minutes
- Format: individual analysis (solo)
- Goal: identify trust-boundary failures and propose defenses
- Deliverable: one-page mitigation brief

Quick Lab: Materials and Safety

- Provided worksheet with sanitized request examples
- No live attack targets, no unauthorized testing
- Defensive analysis only
- Use textbook concepts and chapter vocabulary

Quick Lab Steps (Part A and Part B)

- Part A: classify three samples (parameter tampering, SQLi pattern, path traversal)
- Part B: map each sample to CIA impact and one concrete mitigation
- Assign one owner per mitigation (developer, admin, or policy lead)
- Prepare 60-90 second individual oral summary

Quick Lab Debrief and Rubric

- Accuracy of vulnerability classification
- Quality and feasibility of mitigation
- Correct CIA mapping
- Clarity and professionalism of communication

Career Snapshot: Demand and Compensation

- BLS Information Security Analysts: 29% growth (2024-2034)
- BLS median pay: $124,910 (May 2024)
- BLS projected openings: about 16,400 per year
- CyberSeek U.S. cybersecurity job listings: 514,359 (May 2024-Apr 2025)

Career Framework: NICE Work Roles

- NIST NICE Framework gives common language for cybersecurity work
- Built on Task, Knowledge, and Skill statements
- Useful for job descriptions, curriculum mapping, and skill gap analysis
- Students can map class artifacts to specific role expectations

Credential Pathway for Students

- TEEX AWR173 Information Security Basics (13-hour foundational course)
- ISC2 Certified in Cybersecurity (entry-level cert; exam outline effective Oct 1, 2025)
- CompTIA Security+ SY0-701 (current baseline professional credential)
- PortSwigger Web Security Academy labs for practical evidence

Portfolio Artifacts Employers Understand

- Threat model and mitigation memo from the quick lab
- Scenario timeline and control-gap analysis
- Secure coding checklist tied to Chapter 3 flaws
- Incident-response reflection mapped to NICE tasks

In-Class Writing Reflection (10-12 Minutes)

- Choose one decision from the applied scenario and evaluate alternatives
- Required structure: claim, evidence, counterargument, conclusion
- Connect to one technical control and one ethical principle
- Target length: 250-300 words

Exit Ticket and Next Steps

- Exit Ticket Q1: one concept you can explain confidently now
- Exit Ticket Q2: one concept still unclear
- Exit Ticket Q3: one career step you will take this month
- Next class prep: review notes + chapter exercises for 3/4/5

References (Textbook + Online Supplements)

- Pfleeger, Pfleeger, & Margulies. Security in Computing (5th ed.), Chapters 3-5
- BLS Occupational Outlook: Information Security Analysts
- NIST SP 800-181 Rev.1 (NICE Framework) and NICE Resource Center
- ISC2 Certified in Cybersecurity and exam outline pages
- CompTIA Security+ SY0-701 certification page
- OWASP Top 10:2021 (v1.1 release notice)
- NIST Cybersecurity Framework (CSF 2.0) reference pages
- NIST Secure Software Development Framework (SSDF)
- TEEX AWR173 and PortSwigger Web Security Academy