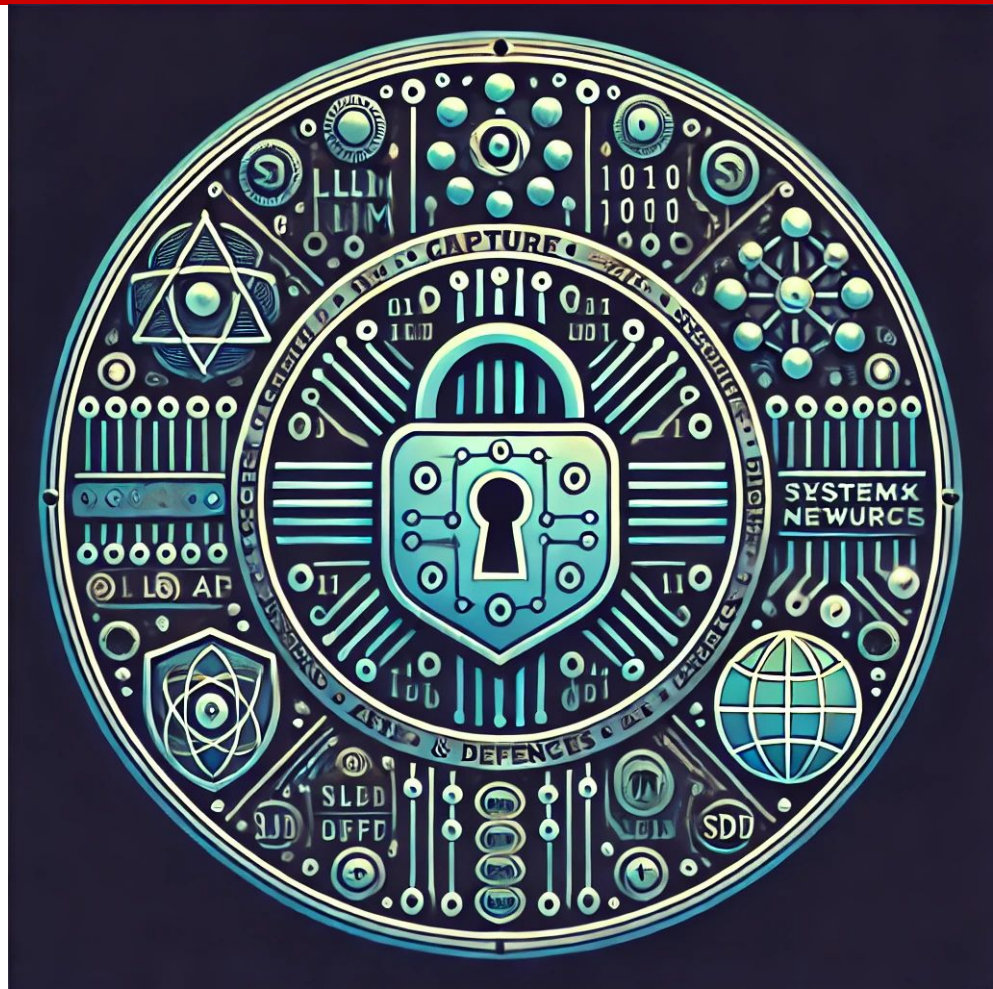
CSC-537 Systems Attacks and Defenses

Vulnerability Management in the Software Supply Chain

Alexandros Kapravelos
akaprav@ncsu.edu



Challenge details

1) Identify vulnerable lines of codes/corresponding files in **challenge.yaml**
2) Make a patched version of your challenge that is not vulnerable. (put a git diff in a patch file)

Introduction to Vulnerability Management

- Explore vulnerability management for components and containers
- Understand its role in software supply chain security
- Learn severity prioritization with CVSS, EPSS, and KEV
- Apply concepts via demos and real-world examples
- Why it matters: trust and security in software delivery

What is the Software Supply Chain?

- Definition
  - Network of entities producing and delivering software
  - Includes developers, vendors, and open-source communities
- Key elements
  - Components: libraries (e.g., Log4j), frameworks (e.g., Spring)
  - Containers: Docker images, Kubernetes pods
- Role in modern development
  - Enables rapid deployment and scalability
  - Introduces security dependencies
- Security and trust: core to user confidence

Components in the Supply Chain

- Types of components
  - Open-source libraries (e.g., OpenSSL, NumPy)
  - Frameworks from repositories (e.g., npm, PyPI)
- Sources
  - Public registries and community contributions
  - Third-party vendors
- Common vulnerabilities
  - Outdated versions with known exploits
  - Malicious code injections
- Maintenance challenge: tracking dependencies

Containers in the Supply Chain

- Definition and purpose
  - Packages apps with dependencies for consistency
  - Used in Docker, Kubernetes environments
- Benefits
  - Portability across platforms
  - Simplified deployment
- Vulnerabilities
  - Vulnerable base images
  - Misconfigured settings (e.g., exposed ports)
- Maintenance: securing runtime environments

Why Vulnerability Management Matters

- Protects software supply chain integrity
  - Prevents breaches from unpatched flaws
  - Maintains trust with users and stakeholders
- Scope
  - Components: libraries and dependencies
  - Containers: images and configurations
- Consequences of neglect
  - Data leaks and system compromises
  - Reputational damage
- Goal: proactive risk reduction

Vulnerability Management Overview

- Systematic process
  - Identify vulnerabilities in components/containers
  - Assess severity and potential impact
  - Prioritize based on risk metrics
  - Mitigate through updates or fixes
- Tools involved
  - Scanners (e.g., `npm audit`, `docker scout`)
  - Databases (e.g., CVE, NVD)
- Outcome: secure and trustworthy software

Step 1: Identification

- Goal: detect vulnerabilities early
- Methods for components
  - Dependency checkers (e.g., `npm audit`)
  - Static analysis tools
- Methods for containers
  - Image scanners (e.g., grype, Trivy, Clair)
  - Runtime monitoring
- Challenges
  - Tracking transitive dependencies
  - Scanning large container images

Step 2: Assessment

- Objective: evaluate vulnerability impact
- Metrics used
  - Severity scores (e.g., CVSS)
  - Exploit likelihood (e.g., EPSS)
- Tools
  - National Vulnerability Database (NVD)
  - Vendor-specific reports
- Outcome
  - Quantified risk for prioritization
  - Contextual understanding of threats

Step 3: Prioritization

- Why prioritize
  - Limited resources for remediation
  - Focus on highest risks first
- Key frameworks
  - CVSS: severity scoring
  - EPSS: exploit probability
  - KEV: known exploited vulnerabilities
- Strategy
  - Combine metrics for efficiency
  - Address critical threats immediately

CVSS: Common Vulnerability Scoring System

- Overview
  - Scores vulnerabilities 0-10
  - Developed by FIRST
- Components
  - Base score: exploitability and impact
  - Temporal score: urgency factors
- Example
  - CVSS 9.8: critical, remote exploit
- Limitation: static severity focus

Notes: CVSS provides a standardized way to assess severity but doesn't predict exploitation likelihood
Refer to CVSS v4.0 User Guide for detailed scoring methodology

EPSS: Exploit Prediction Scoring System

- Purpose
  - Estimates exploit probability (0-1)
  - Maintained by FIRST
- Data sources
  - CVE data and real-world exploits
  - Intrusion detection insights
- Example
  - EPSS 0.9: 90% chance of exploit
- Strength: dynamic threat prediction

Notes: EPSS complements CVSS by focusing on likelihood, using data from honeypots and exploit databases
See EPSS for more

KEV: Known Exploited Vulnerabilities

- Definition
  - CISA-maintained list of exploited CVEs
  - High-priority remediation targets
- Usage
  - Mandated for federal agencies (BOD 22-01)
  - Covers <0.5% of CVEs
- Example
  - Log4j CVE-2021-44228: KEV-listed
- Benefit: immediate action focus

Notes: KEV is a critical resource for supply chain security
Check CISA KEV Catalog for updates

Prioritization Strategy in Practice

- Step-by-step approach
  - Start with KEV-listed vulnerabilities
  - Next, high EPSS scores (>0.8)
  - Then, high CVSS scores (>7.0)
- Benefits
  - Addresses active exploits first
  - Balances severity and likelihood
- Supply chain impact
  - Protects downstream users
  - Maintains trust in delivery

Step 4: Mitigation

- Techniques
  - Patch vulnerable components
  - Update container images
- Best practices
  - Automate updates via CI/CD
  - Test fixes before deployment
- Challenges
  - Dependency conflicts
  - Downtime risks
- Goal: eliminate vulnerabilities

Step 5: Verification

- Purpose
  - Confirm mitigation success
  - Prevent false positives
- Methods
  - Re-run vulnerability scans
  - Penetration testing
- Tools
  - Same as identification (e.g., Trivy)
  - Custom scripts for validation
- Outcome: verified security

Lab Setup: What to Expect

- Two practical demonstrations
  - Component scanning with `npm audit`
  - Container scanning with `docker scan`
- Objectives
  - Identify real vulnerabilities
  - Apply CVSS/KEV prioritization
- Requirements
  - Node.js and Docker installed
  - Sample vulnerable projects

Lab 1: Component Scanning

- Tool: `npm audit`
- Steps
  - Create Node.js project with old `lodash`
  - Run `npm audit` to list vulnerabilities
  - Check CVSS scores in output
- Expected output
  - High-severity issues flagged
  - Remediation suggestions
- Takeaway: proactive dependency checks

Lab 1: Running the Scan

- Command: `npm audit`
- Output details
  - Vulnerability IDs (e.g., CVE-2021-23337)
  - Severity levels via CVSS
- Actions
  - Review affected packages
  - Plan updates (e.g., `npm update`)
- Relevance: catches supply chain risks

Lab 2: Container Scanning

- Tool: `grype`
- Steps
  - Pull an old or known vulnerable image
    - `nginx:1.18`
    - `visiblev8/vv8-base`
  - Scan for vulnerabilities
  - Identify KEV or high CVSS issues
- Expected output
  - List of CVEs with scores
  - Fix recommendations
- Takeaway: secure container deployment

Lab 2: Interpreting Results

- Scan output
  - CVEs with severity ratings
  - Source of vulnerabilities (e.g., base image)
- Prioritization
  - Check KEV status
  - Assess CVSS scores
- Next steps
  - Update to patched image
  - Re-scan to verify

Lab report

https://forms.gle/VcHSrNQYVfWVzYjs5

Template:

Tool output + comments on results

Best Practices: Updates and Scanning

- Regular updates
  - Keep components current
  - Refresh container images
- Vulnerability scanning
  - Integrate into CI/CD pipelines
  - Use multiple tools (e.g., Trivy)
- Frequency
  - Weekly or post-release scans
  - Monitor CVE feeds
- Automation: reduce manual effort

Best Practices: Secure Configurations

- Container hardening
  - Limit privileges (e.g., non-root users)
  - Minimize base image size
- Security measures
  - Avoid exposed secrets
  - Use network policies
- Tools
  - Docker Bench for Security
  - Kubernetes RBAC
- Goal: reduce attack surface

Best Practices: SBOM and Monitoring

- Software Bill of Materials (SBOM)
  - Track all components
  - Enhance visibility
- Continuous monitoring
  - Check for new CVEs
  - Leverage KEV updates
- Tools
  - Syft for SBOM generation
  - Dependabot for alerts
- Outcome: proactive maintenance

Supply Chain Security and Trust

- How maintenance fits
  - Prevents vulnerability propagation
  - Ensures reliable software delivery
- Trust factors
  - User confidence in security
  - Partner reliability
- Risks of failure
  - Breaches affect entire chain
  - Loss of credibility
- Proactive approach: key to success

Tools Recap

- Identification and scanning
  - `npm audit` for components
  - `docker scout/grype` for containers
- Prioritization aids
  - CVSS: severity assessment
  - EPSS: exploit likelihood
  - KEV: exploited threats
- Integration
  - CI/CD for automation
  - SBOM for visibility

Further Learning Resources

- Deep dives
  - CVSS User Guide (first.org/cvss)
  - EPSS Documentation (first.org/epss)
- Official sources
  - CISA KEV Catalog (cisa.gov)
  - NIST SP 800-161 guideline (supply chain)
- Practical tools
  - Trivy, Clair for scanning
  - Dependency-Check for components
- Stay updated: evolving field

Summary and Takeaway Points

- Vulnerability management essentials
  - Maintain components and containers
  - Use CVSS, EPSS, KEV for prioritization
- Supply chain security
  - Proactive fixes ensure trust
  - Neglect risks breaches
- Practical skills
  - Scans with `npm audit`, `grype`
- Call to action: apply in projects