CHAPTER 1

🧠 Chapter Take-Aways (Answer Format)

Q1: Why is software security important?
→ Because attackers often target **software flaws** over hardware or network vulnerabilities. The **app layer is the new battleground**.

Q2: How does security fit into the SDLC?
→ Traditional SDLC often skips security or adds it too late. The **SDL (Security Development Lifecycle)** integrates security into **every phase**.

Q3: What's the difference between quality and security in code?
→ Quality ensures the program works as intended. Security ensures it can't be **abused or exploited**, even when misused.

Q4: What are the goals of the SDL?
→ Enforce **Confidentiality**, **Integrity**, and **Availability** (CIA Triad) while **reducing risk** through secure practices.

Q5: When and how are threat modeling and risk analysis used?
→ Early, ideally during **requirements and design** phases. Used to **anticipate vulnerabilities** before they're built in.

🔐 Core Concepts

- **Software is a strategic asset** – it runs the world (finance, healthcare, government).
- Security flaws are **exploited faster** than ever, often within hours of discovery.
- **Attackers are better organized** and faster than most dev teams.
- SDL is **proactive**, not reactive.

🔄 SDL vs SDLC Chart

Aspect	SDLC	SDL
Focus	Functionality	Security
When security added	Afterthought or at testing	From the beginning
Risk mgmt	Minimal	Integrated
Outcome	Working product	Secure working product

🔐 CIA Triad – The Three Security Goals

Goal	Meaning
Confidentiality	Only authorized access to data
Integrity	Data/system not tampered with
Availability	System/data accessible when needed

🧩 Definitions

Term	Definition
SDL	Security Development Lifecycle – infuses security into all SDLC phases
Threat Modeling	Identifies potential attack paths and how to block them
Risk Analysis	Evaluates the likelihood and impact of potential threats
Attack Surface	All possible ways a system can be exploited
Secure by Design	Building in security from architecture through code

✅ Chapter 1 Quiz (Practice Questions)

1. **What are the three goals of security in SDL?**
   → Confidentiality, Integrity, Availability
2. **How is SDL different from traditional SDLC?**
   → SDL adds security to **each** development phase
3. **When should threat modeling occur?**
   → During the **early stages**, such as design
4. **Why is software the new security battleground?**
   → Most attacks now exploit **software flaws**, not just networks
5. **What does "secure by design" mean?**
   → Security is **built into the architecture**, not bolted on later

🧠 Final Review Tips

- **Security is not quality** – it's protection against misuse.
- SDL must start **as early as requirements**.
- You must know the **CIA triad** cold.
- Understand **why security belongs to all roles**, not just InfoSec.

📘 CHAPTER 1: Introduction (Pages 3–14)

🔑 KEY CONCEPTS

1. Why Software Security Matters

- Software is a **prime target** for attackers because it's often deployed **before it's secured**.
- **Security can’t be patched in later**—it has to be built-in from the start.

2. SDL vs. SDLC

- SDLC = Software Development Life Cycle (traditional dev process)
- SDL = Security Development Lifecycle (adds **security activities** to each SDLC phase)

3. Secure Code ≠ Quality Code

- A program can be "high-quality" but still **insecure** if it doesn't handle threats.

4. The CIA Triad

- **Confidentiality** – Protect data from unauthorized access
- **Integrity** – Ensure data is not tampered with
- **Availability** – Ensure systems are accessible and functional

5. Threat Modeling

- Involves understanding what **assets** need protection, who the **threat actors** are, and how they might attack.

✅ CHAPTER 1 QUIZ QUICK REVIEW

Q: What are the three most important SDL goals?
A: Confidentiality, Integrity, Availability

Q: What's the difference between SDLC and SDL?
A: SDLC is the dev process; SDL integrates **security into every step** of the SDLC.

📘 CHAPTER 2: The Security Development Lifecycle (Pages 15–23)

🔑 TAKEAWAYS

1. Why Traditional Security Fails

- Firewalls, IDS/IPS, and network defenses **don't protect against app-layer attacks** like:
- **XSS (Cross-Site Scripting)**
- **SQL Injection**

- **Buffer Overflows**

2. SDL Is the New Standard

- Fixing vulnerabilities early is **cheaper and more effective**.
- SDL ensures **security is part of design**, not an afterthought.

3. Top SDL Models

- **Microsoft SDL** – Most mature, field-tested
- **BSIMM (Build Security In Maturity Model)**
- **OWASP Code Review Guide**
- **Cisco Secure Development Lifecycle**

4. Benefits of SDL

- Embeds security practices into each phase
- Reduces risk early (before release)
- Encourages collaboration between security and dev teams

🔐 KEY TERMS & DEFINITIONS

Term	Definition
SDL	Security Development Lifecycle – integrates security into every phase of development
SDLC	Software Development Lifecycle – traditional software engineering process
CIA Triad	Confidentiality, Integrity, Availability – core security goals
Threat Modeling	Identifying and analyzing security threats early in the lifecycle
Attack Surface	The sum of all ways a system can be attacked
Cross-Site Scripting (XSS)	Injecting malicious scripts into websites viewed by others
SQL Injection	Attack that manipulates a database through unsafe input handling
Buffer Overflow	Overwriting memory to inject code or crash the system

🧠 TIPS TO PASS THE QUIZ

- Know **why network security is not enough** (apps are the new attack vector).
- Be able to explain the **difference between SDL and SDLC**.
- Understand the **benefits of using SDL early in dev**.
- Memorize the **CIA Triad** and what each goal protects.

CHAPTER 2

🎯 Chapter Take-Aways (with Answers)

Q1: What are the biggest challenges in making software secure?
→ Resource limits, lack of skilled talent, tool gaps, and resistance to process change.

Q2: What models help guide secure development maturity?
→ BSIMM, SAMM, and ISO/IEC 27034.

Q3: What tools and roles are critical to software security?
→ Static/dynamic scanners, threat modeling tools, fuzzers. Roles include secure devs, architects, champions.

Q4: Why are principles like least privilege important?
→ They reduce risk exposure and limit attack surfaces.

Q5: How does SDL map to SDLC and Agile/Waterfall?
→ SDL wraps around any dev model. Security tasks are inserted into each SDLC phase.

🔐 2.1 Overcoming Challenges

- Software is complex → more bugs.
- Lack of **built-in** security → patchwork solutions.
- SDL aims to be **repeatable**, **measurable**, and **cost-effective**.

🏗️ 2.2 Software Security Maturity Models

- **BSIMM (Building Security In Maturity Model)**: Observational, used to **benchmark** orgs.
- **SAMM (Software Assurance Maturity Model)**: **Prescriptive**—helps you build a roadmap.
- **ISO/IEC 27034**: Global standard for app security practices.

🧠 Tip: BSIMM = observe, SAMM = do.

📚 2.3 – 2.4 SDL Best Practice Resources

- **SAFECode**: Industry-led security dev guidance
- **DHS Software Assurance Program**: US gov't recommendations
- **NIST**: Authoritative security controls
- **CVE**: Database of known vulnerabilities
- **SANS Top 25**: Most common coding errors
- **DoD CSIAC**: Secure development resources
- **CERT/Bugtraq/SecurityFocus**: Real-world threat info

🛠️ 2.5 Critical Tools and Talent

Tools:

- **Static analysis** (SAST): finds bugs without executing code
- **Dynamic analysis** (DAST): tests running apps
- **Fuzzing**: bombards app with input to crash it
- **Threat modeling** tools like Microsoft's Threat Modeling Tool

Talent:

- **Security architects**
- **DevSecOps engineers**
- **Security champions** in each dev team

🔒 2.6 Least Privilege

- Principle: **only grant what's necessary**
- Minimizes the impact of a compromise
- Essential for **RBAC** and Zero Trust

🔐 2.7 Privacy

- Protect **PII** by:
- Minimizing collection
- Encrypting at rest & in transit
- Enforcing access controls
- Must align with:
- GDPR, CCPA, HIPAA
- Privacy by design!

📊 2.8 Importance of Metrics

Metrics = proof of progress.

Examples:

- % of code covered by static scans
- of security issues fixed before release
- of secure coding training completions

🔁 2.9 SDL to SDLC Mapping

SDLC Phase	SDL Activity
Requirements	Threat modeling, privacy planning
Design	Architecture analysis, secure patterns
Implementation	Static code analysis, coding standards
Testing	DAST, fuzzing, pen testing
Deployment	Security review, license checks
Maintenance	Patch mgmt, vulnerability disclosure

SDL is flexible — works with **Waterfall, Agile, DevOps**.

🔧 2.10 Software Development Methodologies

Waterfall:

- Linear, phase-based
- Security fits **neatly** into each phase
- Good for **strictly regulated environments**

Agile:

- Iterative
- Security must be **embedded** in sprints
- Requires **automation** and culture shift

✅ Chapter Quick Review

Question	Answer
What is SDL?	A security-focused lifecycle process
Which maturity models exist?	BSIMM, SAMM, ISO/IEC 27034
What are key SDL tools?	SAST, DAST, fuzzers, threat modeling
Why use least privilege?	Minimizes risk exposure
What's needed for privacy?	Encryption, access control, legal compliance

📖 Terms to Know

Term	Definition
BSIMM	Measures real-world SDL practices
SAMM	Prescriptive roadmap for building SDL
Least Privilege	Users/systems get minimum access needed
Fuzzing	Sending random data to crash/find bugs
CVE	Public list of known software vulnerabilities

🔑 CHAPTER TAKE-AWAYS

- Understand **process, people, and tech** challenges to secure software.
- Compare **maturity models** for SDL effectiveness.
- Review industry standards (e.g., **ISO/IEC 27034**, **SAFECode**, **NIST**, etc.).
- Learn the importance of **least privilege**, **privacy**, **metrics**, and **mapping SDL to SDLC**.
- Know common development models like **Waterfall** and **Agile** and how to secure them.

📍 SECTION BREAKDOWN & CORE CONCEPTS

✅ 2.1 Overcoming Challenges in Making Software Secure

- Legacy mindset: “Network security is enough” = WRONG
- Application-layer attacks bypass traditional defenses:
- XSS, SQL Injection, Buffer Overflow
- SDL provides:
- **Integrated security controls**
- Prevents & mitigates **vulnerabilities at the code level**
- **Takeaway**: SDL must be built-in, not bolted on.

✅ 2.2 Software Security Maturity Models (SSMM)

Maturity models measure how advanced your security practices are.

- **BSIMM** (Build Security In Maturity Model)
- Tracks real SDL practices across 100s of organizations.
- **OpenSAMM** (OWASP)
- Helps orgs build and measure software security initiatives.
- **Microsoft SDL**
- Most referenced; used to secure Windows/Office

📌 All models stress:

- Early threat modeling
- Policy enforcement
- Developer training
- Secure coding standards

✅ 2.3 ISO/IEC 27034 – Application Security

- International standard for **securing software** in organizations.
- Core goals:
- **Integrate security into business functions**
- Focus on **Application Security Controls (ASCs)** — reusable secure components

✅ 2.4 Resources for SDL Best Practices

Organization	**Role**
SAFECode	Promotes secure coding practices
DHS Software Assurance Program	Guides on security controls
NIST	SDL mapping, secure SDLC guidelines (SP800-64)
MITRE/CVE/NVD	Vulnerability tracking and scoring (CVSS)
SANS Top 20	Lists top cybersecurity risks
CSIAC (DoD)	Defense-focused SDL knowledge
CERT, Bugtraq, SecurityFocus	Early vulnerability alerts & exploit DBs

✅ 2.5 Critical Tools & Talent

Tools:

- **Static & dynamic analyzers** (e.g., SonarQube, Fortify)
- **Fuzzers, threat modelers, penetration testers**

Talent:

- **Security Champions** inside dev teams
- Skilled coders with secure design knowledge
- Analysts who understand **business risk + tech**

✅ 2.6 Principle of Least Privilege (PoLP)

- Users, processes, and systems should have **ONLY the permissions they need**.
- Reduces blast radius when attacks occur
- Must be **designed into** software—not an afterthought

✅ 2.7 Privacy

- Ensure SDL includes **privacy-by-design**
- Add **data minimization**, **encryption**, **access control**, **user consent**
- SDL tools can help enforce **PIA (Privacy Impact Assessments)**

✅ 2.8 Importance of Metrics

- You can't improve what you can't measure:
- **# of vulnerabilities discovered per phase**
- **Time to remediate**
- **# of secure code training sessions completed**
- Metrics help **prove ROI** of SDL

✅ 2.9 Mapping SDL to SDLC

SDLC Phase	SDL Activity
Requirements	Identify security & privacy goals
Design	Threat modeling
Implementation	Secure coding & static analysis
Testing	Pen testing, dynamic scans
Deployment	Review configs & release checklists
Maintenance	Patch management, metrics tracking

✅ 2.10 Software Development Methodologies

2.10.1 Waterfall

- Sequential (no going back)
- SDL activities must be **front-loaded** during design

2.10.2 Agile

- Fast-paced, iterative
- SDL must be **modularized** (e.g., small secure coding tasks per sprint)
- Use **automation** for frequent tests

🧠 KEY TERMS (MEMORIZE THESE)

Term	Definition
SDL	Security Development Lifecycle – secure coding and processes integrated into SDLC
BSIMM	Maturity model to benchmark software security programs
OpenSAMM	OWASP's model to build measurable SDL
ISO/IEC 27034	International SDL standard for app security
CVE/NVD	Common vulnerability registries & scoring (CVSS)
Principle of Least Privilege	Give only minimum permissions required
PIA	Privacy Impact Assessment
Secure Coding	Techniques to avoid injection, buffer overflow, etc.

✅ QUIZ PREP: SAMPLE QUESTIONS

1. **Which model helps measure SDL maturity?**
   → **BSIMM or OpenSAMM**
2. **What standard defines application-level security controls?**
   → **ISO/IEC 27034**
3. **Which phase aligns with threat modeling in SDL?**
   → **Design**
4. **Why is least privilege important?**
   → **It limits damage from exploited components**
5. **What is one advantage of integrating SDL into Agile?**
   → **Security tasks become part of each sprint**

📝 QUICK REVIEW SHEET

- SDL must be **embedded in the SDLC**
- **Maturity models** (BSIMM, OpenSAMM) help assess and grow
- **ISO/IEC 27034** is key standard for secure app design
- Tools + Talent = SDL success
- Apply **least privilege**, **privacy by design**, **metrics**
- **Agile** = frequent secure coding checks
- **Waterfall** = upfront secure design

CHAPTER 3

🎯 Chapter Take-Aways (with Answers)

Q1: Why involve the security team early in a project?
→ To identify risks, shape security requirements, and influence design before costly mistakes happen.

Q2: What happens in a discovery meeting?
→ The team gathers app details (purpose, data, architecture), sets security scope, and identifies stakeholders.

Q3: What is an SDL Project Plan?
→ A roadmap that defines **security milestones**, owners, artifacts, and key activities across the SDLC.

Q4: What's the role of a PIA (Privacy Impact Assessment)?
→ Ensures **privacy risks are known** early and remediated during design — especially when handling PII.

Q5: What defines success in the A1 phase?
→ Clear planning, early engagement, risk visibility, and measurable artifacts.

🔁 Key A1 Activities

✅ *3.1 Involve Security Team Early*

- Security isn't a last-minute task.
- Bring in the software security team **before design starts**.
- Benefits:
- Threat modeling ready early
- Fewer bugs = cheaper fixes
- Security woven into the system architecture

🧠 *3.2 Host a Discovery Meeting*

- Participants: Architects, developers, security team, product owners.
- Goals:
- Define the app's **functionality** and **data sensitivity**
- Review architectural decisions
- Identify third-party components
- Understand legal/privacy obligations
- Output: **Security posture baseline** and list of known gaps

📋 *3.3 Create the SDL Project Plan*

- Living document = adjusts as project changes
- Includes:
- Security milestones (threat model, pen test)
- Roles/responsibilities
- Tooling (SAST/DAST, PIA)
- Deliverables (risk register, test plan, logs)
- Helps track **compliance and accountability**

🕵️ *3.4 Launch the Privacy Impact Assessment (PIA)*

- Mandatory if handling:
- **PII**, **PCI**, **PHI**, etc.
- Identifies:
- What data is collected/stored/shared
- Who accesses it
- Applicable laws (GDPR, HIPAA, etc.)
- Privacy by design starts **right here**

📊 3.5 A1 Key Success Factors & Metrics

🔑 *3.5.1 Success Factors*

- Clear communication with dev team
- Documented security requirements
- Effective cross-functional collaboration
- Use of **standardized** security templates/checklists

📦 *3.5.2 Deliverables*

- SDL project plan
- Discovery meeting notes
- Risk register
- Initial threat model
- PIA initiation record

📈 *3.5.3 Metrics*

- of projects with early security engagement
- % of apps with completed SDL plan
- PIA coverage ratio
- Threat models completed vs. required

✅ Chapter Quick Review

Question	Answer
Why include security early?	Lower cost, more secure design
What is discovery meeting for?	Understand app, scope risk, collect context
What's in SDL project plan?	Milestones, owners, security activities
Why use a PIA?	Identify & mitigate privacy risk early
A1 metrics?	Security engagement %, threat models completed, etc.

📖 Key Terms Defined

Term	Definition
Discovery Meeting	Early discussion of architecture, scope, and data
SDL Project Plan	Document that tracks security steps across SDLC
PIA	Assessment of how personal data is handled and protected
Security Posture	Current state of security readiness and risk exposure
Risk Register	List of known security threats and planned mitigations

📍 2.10.1 Waterfall Development (SDL Integration in Waterfall)

- **Waterfall** = linear, stage-by-stage development
- SDL tasks must be embedded **upfront**
- Planning and early **threat modeling** is critical
- **Risk**: Security missed if you wait until testing or deployment

🧠 Key SDL Add-ons for Waterfall:

- Add **security gates** between stages
- Require **design reviews** for security before coding
- Focus on **early policy compliance** and **secure requirements gathering**

📍 2.10.2 Agile Development (SDL Integration in Agile)

- Agile is **iterative** and **time-boxed**
- Security must be **modular** and **automated**
- Security tasks = small, manageable per sprint
- Use **automation tools**: Static Analysis, Dynamic Analysis, etc.

🧠 Key Practices:

- Threat modeling must evolve with each sprint
- Use a **security backlog**: logged risks, mitigations, and fixes
- Integrate **security test cases** in unit/integration testing

🧩 SDL in Agile vs Waterfall

Feature	Waterfall	Agile
Security timing	Front-loaded	Sprint-by-sprint
Threat modeling	Once during design	Ongoing, updated
Testing	End of cycle	Throughout development
SDL tools	Manual + phased	Automated + continuous
Flexibility	Rigid	High, requires DevSecOps

📝 2.11 Chapter Summary

- Security must fit the **dev methodology** used
- Waterfall: heavy **early investment** in SDL planning
- Agile: continuous **automation and integration**
- Successful SDL adapts to your **people, process, and tech**

📘 Chapter 3 Begins – Security Assessment (A1)

✅ 3.1 Software Security Team Looped In Early

- A1 = Discovery Phase
- Identify **product risk profile**
- Outline SDL activities and controls
- Sync with project schedule

🎯 **4 Key Questions:**

1. What is the software supposed to do?
2. What assets need protection?
3. What laws/regulations apply?
4. What are the threats?

✅ 3.2 Discovery Meeting

- Security team meets with:
- Business owners
- Developers
- Compliance officers

📌 Goal: Understand **compliance**, **privacy**, **data handling**, and **threats** early.

Deliverables:

- Asset inventory
- Risk classification
- Regulatory impact mapping
- Potential third-party risk

✅ 3.3 SDL Project Plan Created

- A living document:
- SDL milestones
- Required security activities
- Assigned responsibilities
- Mapped to SDLC timeline

Why it matters:

- Keeps SDL **visible & measurable**
- Forces integration into planning

✅ 3.4 Privacy Impact Assessment (PIA)

- Required if software handles **PII (Personal Identifiable Information)**
- Helps meet **legal and regulatory standards**
- Considers:
- Data minimization
- Access control
- Data encryption
- Consent and user rights

🧠 KEY TERMS TO REMEMBER

Term	**Definition**
Agile SDL	Modular, security-by-sprint integration with automation
Waterfall SDL	Front-loaded security embedded into early stages
Discovery Phase (A1)	First SDL phase to assess risk, compliance, and outline plan
SDL Project Plan	A document mapping SDL steps to SDLC stages
PIA (Privacy Impact Assessment)	Evaluation of how PII is handled securely and lawfully

📘 FINAL EXAM PRACTICE QUESTIONS

5. **In Waterfall, when should SDL activities begin?**
   → At the **beginning**, during planning/design

6. **What makes SDL work in Agile?**
   → Integration into **sprints**, use of **automation**, and evolving **threat models**

7. **What's the first goal in the Security Assessment phase?**
   → Define the **product's risk profile**

8. **Who attends the Discovery Meeting?**
   → Business stakeholders, devs, legal/compliance, and **security team**

9. **What does a PIA ensure?**
   → **Privacy compliance** and proper **PII handling**

CHAPTER 4

🏛️ CHAPTER 4 – Architecture (A2): SDL Activities and Best Practices

Q1: What is the A2 phase about?
→ Ensures software architecture considers **security, privacy, and compliance** early, based on business risk.

Q2: What's the purpose of threat modeling?
→ Identify **entry points**, **data flows**, and **threats** to reduce security issues in architecture.

Q3: Why analyze open-source components?
→ Prevents **vulnerabilities and license issues** from entering through third-party code.

Q4: What does Privacy Information Gathering do?
→ Helps design privacy-respecting systems by **mapping how PII is used**, stored, and shared.

Q5: What defines success in A2?
→ Effective threat modeling, policy compliance, and documented architecture-level security decisions.

 4.1 Policy Compliance Analysis

- Review and **map internal SDL policy** to app features.
- Identify **external regulations**: HIPAA, GDPR, PCI, etc.
- Goal: Bake security/privacy expectations into design before coding starts.

📐 4.2 SDL Policy Assessment and Scoping

- Identify which **security and privacy policies** apply.
- Confirm with stakeholders the scope of:
- Risk tolerances
- Data classifications
- Control expectations
- Results feed into architectural **security decisions**.

🕵️ 4.3 Threat Modeling & Architecture Security Analysis

🔍 *4.3.1 Threat Modeling*

- Focus: Understand potential **attacker strategies**
- Use STRIDE (Spoofing, Tampering, Repudiation, Info Disclosure, Denial, Elevation)
- Consider abuse cases, threat trees, misuse cases

🔄 *4.3.2 Data Flow Diagrams (DFDs)*

- Identify:
- **Trust boundaries**
- Input/output channels
- Sensitive data paths
- Enables tracking **attack surfaces** clearly

🧠 *4.3.3 Architectural Threat Analysis*

- Analyze design choices:
- APIs
- Frameworks
- Communication flows
- Rank risks using CVSS or DREAD

🧯 *4.3.4 Risk Mitigation*

- Design in:
- Input validation
- Logging
- Authentication mechanisms
- Build **secure-by-design** patterns into architecture

🌐 4.4 Open-Source Selection

- Assess OSS libraries for:
- **Security history**
- **License compatibility**
- Community activity
- Keep **SBOM** (software bill of materials)

🤲 4.5 Privacy Information Gathering

- Identify:
- What PII is collected?
- Where is it stored/sent?
- Who has access?
- Result: Enables "privacy by design"

📊 4.6 Key Success Factors & Metrics

✅ *4.6.1 Key Success Factors*

- Involve all stakeholders
- Maintain traceable records
- Use standard threat models & DFD templates
- Update architecture with evolving threats

📦 *4.6.2 Deliverables*

- Completed threat models
- Architectural review documents
- Privacy data maps
- Open-source component inventory
-

📈 *4.6.3 Metrics*

- % of projects with completed threat model
- of high-risk architecture flaws detected
- Time to complete policy scoping

✅ Chapter Quick Review

Question	**Answer**
Purpose of A2 phase?	Ensure secure architecture decisions
What is STRIDE?	Threat modeling categories
Why DFDs matter?	Visualize data flows/trust zones
OSS risk?	License + security vulnerabilities
Privacy info gathered?	What/where/who handles PII

📖 Key Terms Defined

Term	**Definition**
Threat Modeling	Process to identify and rank threats
Data Flow Diagram (DFD)	Visual map of how data moves
Open Source Review	Vetting 3rd-party code for security & license issues
Risk Mitigation	Security controls added at design level
PII Map	Diagram of personal data handling in the system

✅ *3.5 Key Success Factors and Metrics*

🔷 3.5.1 Key Success Factors:

- Clearly identified **stakeholders** and their **security responsibilities**
- Minimum **security** and **privacy** requirements documented
- SDL Plan finalized early in the **SDLC**
- Implementation of a **tracking system** for vulnerabilities (like JIRA)

✅ Tip: Think of these as the "ground rules" that set the project up for success.

🔷 3.5.2 Deliverables from A1 Phase:

1. **Stakeholder matrix**
2. **Security roles/responsibilities**
3. **Minimum security requirements**
4. **PIA (Privacy Impact Assessment) outline**
5. **SDL work item tracking** setup
6. **Threat profile** — inputs for next phases like threat modeling

🔷 3.5.3 Metrics:

Track these during A1 to gauge maturity:

- of stakeholders assigned security roles
- Time to complete PIA
- of security controls identified
- Ratio of unresolved A1 tasks before A2

✅ Metrics = Evidence that security was "built-in" from Day 1

✅ *3.6 Summary – A1 Best Practices*

- Security should be **part of planning** not post-release
- Privacy and security risk assessments are **mandatory**
- SDL foundation set during A1 = smoother secure dev cycle

✅ *Chapter Quick-Check (Sample Questions)*

7. **Why is the Discovery Meeting important?**
   → To integrate **security from the start** of development
8. **PIA ensures?**
   → Lawful and secure handling of **PII (Personal Identifiable Information)**
9. **Threat profile created in A1 feeds into?**
   → **Requirements gathering** and **threat modeling**

🧱 CHAPTER 4 BEGINS – Architecture (A2) Phase

🧩 *4.0 Chapter Overview*

- A2 ensures **security requirements**, **threats**, and **constraints** are accounted for in architecture.
- It includes:
- **Policy reviews**
- **Threat modeling**
- **Risk ranking**
- **Design constraints**
- **Open-source code review**

✅ Focus: Embed security early in the **technical blueprint** of the system

✅ *Chapter 4 Take-Aways (Preview)*

- Learn how to apply **policy compliance checks**
- Perform effective **threat modeling** using Data Flow Diagrams (DFDs)
- Identify and rank **architectural threats**
- Evaluate **open-source components** for risk
- Prepare security-focused architecture deliverables

🧠 KEY TERMS TO MEMORIZE

Term	**Definition**
PIA	Privacy Impact Assessment – ensures lawful use of personal data
Threat Profile	Document listing threats, actors, and vectors for a system
SDL Plan	Roadmap aligning SDL activities with SDLC phases
Security Roles Matrix	Identifies who owns what security responsibility
A1 Phase	Initial SDL phase where risk, privacy, and planning is done

📝 EXAM REVIEW SHEET (Pages 53–64)

Question	**Answer**
What is the goal of the A1 phase?	Establish foundation for secure development
What does the SDL Plan contain?	Milestones, controls, roles, requirements
Why is tracking security items important?	For accountability and auditability
Which phase does A1 feed into?	A2 – Architecture
Why do metrics matter in A1?	Proves maturity and process discipline

CHAPTER 5

✅ CHAPTER 5 – Design and Development (A3): SDL Activities and Best Practices

📌 Chapter Take-Aways

- Understand **A3 phase activities** in the SDL.
- Learn **key success factors** for A3.
- Create **draft deliverables and a test plan** for a case study.
- Perform **threat model updates**, design reviews, and privacy assessments.

🔍 Key Concepts

5.1 A3 Policy Compliance Analysis

- Continues from A2's policy review.
- Includes external policies: corporate, privacy, open-source, and IP protection.
- **Goal**: Ensure developers follow external and internal rules for secure and private code.

5.2 Security Test Plan Composition

- Test plans should align with functional/design specs.
- Must support:
- **Security requirements validation**
- **Threat mitigation validation**
- **Privacy controls**
- Should include: security roles, attack surface, and test coverage matrix.

5.3 Threat Model Updating

- Review/update threat models from A2.
- Analyze for new design risks or updates from requirement changes.
- Include **architectural reviews** and **attack surface updates**.

5.4 Design Security Analysis and Review

- Reviews that verify secure design:
- Proper authentication/authorization.
- Secure session management.
- Use of approved crypto libraries.
- Input validation.
- Tools: checklists, STRIDE, architectural risk analysis.

5.5 Privacy Implementation Assessment

- Examine:
- Data collection/storage practices.
- Access controls on personal data.
- Compliance with privacy policies.
- Verify privacy features match business/legal requirements.

🧠 Key Success Factors (5.6.1)

- All tests and designs mapped to security/privacy requirements.
- Updated threat model and attack surface documentation.
- Defined secure coding best practices.

📄 Deliverables (5.6.2)

- Security Test Plan.
- Updated threat model.
- Secure Design Review document.
- Privacy Assessment.

📊 Metrics (5.6.3)

- % of code reviewed for security/privacy.
- of mitigated threats or vulnerabilities.
- Test coverage for security and privacy controls.

🔐 Key Terms

Term	Definition
A3	SDL Phase focused on secure design and development
Security Test Plan	Document defining test cases for security validation
Threat Model	Structured analysis of threats and vulnerabilities
Privacy Assessment	Evaluation of how personal data is handled securely
Secure Coding Standards	Rules to prevent introducing vulnerabilities in code

🧠 Summary

The A3 phase sets the foundation for a **secure, privacy-aware software build**. It's all about:

- Aligning design with security/privacy policies.
- Updating and verifying threat models.
- Planning test coverage.
- Ensuring privacy protections are implemented *before* code is written or shipped.

📝 Quiz Questions (Practice)

1. **What is the main goal of the A3 policy compliance analysis?**

- Ensure compliance with internal and external security/privacy policies.

2. **Why is threat model updating necessary in A3?**

- To capture new risks introduced during design or requirement changes.

3. **What should a security test plan include?**

- Roles, attack surface, test types, and mapping to requirements.

4. **What is verified during design security reviews?**

- Correct use of auth, input validation, crypto libraries, and secure storage.

5. **What's the purpose of a privacy implementation assessment?**

- Validate data protection practices and legal compliance.

🧱 CHAPTER 5 – DESIGN & DEVELOPMENT (A3) SDL PHASE

🎯 Chapter Objectives (Take-Aways)

- Analyze design and code for **security policy compliance**
- Develop a **security test plan**
- Perform **threat model updates**
- Conduct a **design security review**
- Assess **privacy implementation**
- Document success factors, deliverables, and metrics

🧩 5.1 A3 Policy Compliance Analysis

- Ensure architecture/design aligns with:
- SDL goals
- Regulatory & business security policies
- Focus Areas:
- Authorization, authentication, auditability
- Compliance frameworks (HIPAA, GDPR, PCI, etc.)
- Secure architectural patterns (zero trust, defense in depth)

🔍 Example: Ensure authentication uses **secure protocols** (e.g., OAuth2, SAML)

🧪 5.2 Security Test Plan Composition

- Plan defines **how, what, and when** to test:
- Functional security (e.g., access control)
- Non-functional security (e.g., performance under attack)
- Include:
- **Static Analysis (SAST)**
- **Dynamic Analysis (DAST)**
- **Pen Testing**
- **Fuzz Testing**
- Regression test plans for security fixes

🧠 Tip: Think of this as the **blueprint for secure QA**

🔄 5.3 Threat Model Updating

- Architecture and code changes? → Update threat model
- Keep threat models **aligned with reality** (new services, APIs, features)
- Include abuse cases & potential privilege escalation

🛡️ Use tools like:

- STRIDE (Spoofing, Tampering, Repudiation, Info Disclosure, DoS, Elevation)
- DFDs (Data Flow Diagrams)

🔍 5.4 Design Security Analysis and Review

- Validate that design decisions meet:
- Security requirements
- Privacy controls
- Risk mitigation expectations
- Techniques:
- Whiteboard reviews
- Architecture walkthroughs
- Peer code review (esp. for secure libraries)

✅ Look for:

- Least privilege enforcement
- Secure session management
- Secure data storage and transmission

🔐 5.5 Privacy Implementation Assessment

- Check if system:
- Minimizes data collection
- Applies encryption to PII
- Implements consent and access controls
- Ensure **privacy by design** — not after the fact
- Aligns with PIA (Privacy Impact Assessment) created earlier

🧠 Watch for data leaks, excessive logging, or insecure defaults

🏆 5.6 Key Success Factors and Metrics

🧱 5.6.1 Success Factors

- Architecture mapped to security requirements
- Updated threat models reviewed
- Secure design patterns reused
- Security test plan in place **before implementation**

📦 5.6.2 Deliverables

- Updated **threat model**
- Completed **design security review**
- Documented **privacy compliance status**
- Finalized **security test plan**

📊 5.6.3 Metrics

Track:

- % of design flaws caught in review
- of critical vulnerabilities before code complete
- Time to update threat model after design changes

✅ CHAPTER 5 QUICK REVIEW

Question	**Answer**
What's the main task in A3?	Secure design & test planning
Why update threat models?	Code & architecture change
What is in a security test plan?	SAST, DAST, fuzz, pen testing
What does the privacy review look for?	Data minimization, access control, encryption
What proves A3 success?	Design flaws caught early, test coverage, secure reviews

🧠 TERMS TO KNOW

Term	**Definition**
Threat Modeling	Process to identify, analyze, and mitigate threats
Security Test Plan	Blueprint for how security testing is conducted
Design Security Review	Validation of architecture/design against security policies
Privacy by Design	Embedding privacy from initial design phases
Static/Dynamic Analysis	Code scanning before & during execution

CHAPTER 6

✅ CHAPTER 6 – Design and Development (A4): SDL Activities and Best Practices

- Understand the *Design and Development (A4)* phase of the Security Development Lifecycle (SDL)
- Learn how to conduct *security test case execution*
- Use tools like *static, dynamic, and fuzz testing*
- Conduct *manual code reviews*
- Know *privacy validation* and *remediation* techniques
- Prepare testing reports for security compliance

🔑 **MAJOR CONCEPTS & DEFINITIONS:**

Term	Definition
A4 Phase	The "readiness" stage in the SDL. Prepares software for secure release.
Static Analysis	Examines source code before compiling to catch vulnerabilities early.
Dynamic Analysis	Runs the application and watches for misbehavior during execution.
Fuzz Testing	Sends malformed/random data to crash app or reveal input vulnerabilities.
Manual Code Review	Human-led inspection of source code to detect security flaws.
Policy Compliance Analysis	Ensures that the application follows org's security policies at this stage.
Security Test Case Execution	Run a defined set of security tests for different scenarios and outcomes.

🧠 KEY ACTIVITIES IN A4 PHASE

1. **Policy Compliance Analysis (6.1)**

- Validate project complies with internal security policies.
- Ensure alignment with SDL documentation.

2. **Security Test Case Execution (6.2)**

- Develop security-focused test cases.
- Measure against known vulnerabilities.
- Includes both *positive* and *negative* test scenarios.

3. **Code Review (6.3)**

- Use automated tools and manual reviews.
- Helps catch:
- Unsafe function calls
- Poor error handling
- Missing input validation

4. **Security Tools (6.4)**

- **Static Analysis** (early detection): syntax, API usage, logic flaws.
- **Dynamic Analysis** (run-time testing): memory leaks, race conditions.
- **Fuzz Testing** (crash it!): test for unpredictable input/output behavior.
- **Manual Code Review** (human + tool insights).

🧩 KEY SUCCESS FACTORS (6.5)

- Integration of security tools into the dev process.
- Frequent code reviews and test cycles.
- Timely remediation of identified issues.
- Updated documentation after test cycles.

📦 DELIVERABLES (6.6)

- Security testing results
- Updated test cases
- Security compliance status reports
- Issue tracking and remediation documentation

📊 METRICS (6.7)

- of test cases executed
- of issues discovered
- % compliance with security requirements
- of retests done
- Severity levels of open issues

📚 CHAPTER SUMMARY (6.8)

This phase builds assurance through testing. You verify the product is secure using both **automated tools** and **manual reviews**. It's about making sure vulnerabilities are identified, tested, and resolved before the next phase (A5 Ship). Testing and validation are continuous.

CHAPTER 7

📘 Chapter 7 – **Ship (A5): SDL Activities and Best Practices**

🎯 Chapter Take-Aways (Test Focus)

- Understand the final activities before shipping secure software.
- Conduct final **SDL policy compliance analysis**.
- Execute **vulnerability scans** and **code-assisted penetration testing**.
- Perform **open-source licensing reviews**.
- Complete **final security and privacy reviews**.
- Know the **key success factors**, **deliverables**, and **metrics** for the shipping phase.

Key Concepts and Processes

7.1 A5 Policy Compliance Analysis

- Final check to ensure **all SDL policies** are applied.
- Tailored by product type, code type, platform, and associated risks.

7.2 Vulnerability Scan

- Automated tools scan for known vulnerabilities.
- Ensures software and surrounding systems are clean before release.

7.3 Code-Assisted Penetration Testing

- Goes beyond scanning: simulates real-world attacks.
- Involves a **third-party security expert/team** ("another set of eyes").
- Provides an **independent, deep-dive security validation**.

7.4 Open-Source Licensing Review

- Checks all third-party and open-source software used.
- Ensures legal and compliance requirements are met.
- Prevents unintentional violations that could cause lawsuits or revocation of usage rights.

7.5 Final Security Review

- Aggregates outputs of all SDL activities: threat models, scan results, security tests.
- Verifies if security requirements from earlier phases were met.

7.6 Final Privacy Review

- Ensures privacy concerns are addressed before shipping.

- Includes compliance with **data protection laws (like GDPR, HIPAA, etc.)**.

⭐ 7.7 Key Success Factors

- **Early and continual involvement** of security teams.
- Complete and accurate documentation.
- Effective collaboration between **legal**, **engineering**, and **security** teams.

📦 7.8 Deliverables

- Final **security and privacy reports**
- Documentation of scans, test results, SDL compliance checklists
- Review sign-offs from all stakeholders

📊 7.9 Metrics

- Number of open vs. resolved vulnerabilities
- Penetration test findings
- Licensing compliance success rate
- Time spent fixing final-phase security issues

🧠 7.10 Summary

- This phase is the **final gate before release**.
- If any **critical issues remain unresolved**, the product may not be cleared for release.
- Includes a **go/no-go decision** based on security and privacy review outcomes.

✅ Chapter Quick-Check (Practice Quiz)

Q1: What's the purpose of the final SDL policy compliance analysis?
A: Ensure all security and privacy requirements are met for release based on project type and platform.

Q2: What's the difference between vulnerability scanning and penetration testing?
A: Scanning is automated and detects known flaws; penetration testing simulates real attacks manually or semi-automatically.

Q3: Why is a third-party used for penetration testing?
A: Provides an independent perspective and deeper evaluation.

Q4: What does the final privacy review check for?
A: Ensures data handling complies with legal and internal privacy requirements.

Q5: What is a common metric tracked in the ship phase?
A: Ratio of resolved to unresolved vulnerabilities.

📜 Bold Terms & Definitions

Term	Definition
SDL Policy Compliance	Ensures security guidelines are followed throughout the lifecycle.
Vulnerability Scan	Automated search for known security weaknesses.
Penetration Test	Manual or semi-automated simulated attack to find vulnerabilities.

Open-Source Licensing Review	Checks for proper usage and legal compliance of third-party code.
Final Security Review	Aggregated review of all security activities to determine readiness.
Final Privacy Review	Ensures personal data handling meets privacy requirements.

CHAPTER 8

✅ CHAPTER 8: Post-Release Support (PRSA1–5)

- Understand the structure and role of post-release support in software security.
- Learn how to build a responsive PSIRT (Product Security Incident Response Team).
- Grasp third-party review, post-release certification, and legacy/M&A security assessment processes.
- Identify key deliverables, metrics, and organizational roles for ongoing support.

🔗 **Key Sections & Summary:**

📌 ***8.1 Right-Sizing the Software Security Group***

- Needs a **minimum of one principal security architect**.
- Larger orgs: SSCs (Security Champions) per product + SSEs (Evangelists) for training and policy enforcement.
- Ensure software teams are empowered to own their own security.

📌 ***8.2 PRSA1: External Vulnerability Disclosure Response***

- Define a process for post-release vulnerabilities.
- Roles: PSIRT, privacy teams, legal, comms, dev teams.
- Use bug bounty programs & vulnerability databases (e.g., CERT, NVD).

📌 ***8.3 PRSA2: Third-Party Reviews***

- External, independent assessments.
- 2 versions of each report: internal (detailed), external (sanitized).
- Should be comprehensive and include remediation recommendations.

📌 ***8.4 PRSA3: Post-Release Certifications***

- Driven by market, customer, or regulatory demand.
- Requires periodic renewal if conditions persist (e.g., SOC 2, FedRAMP).

📌 ***8.5 PRSA4: Internal Review for New Product Combos or Cloud Deployments***

- Triggered when combining products or moving to cloud.
- Needs new risk/threat model analysis.
- Apply same SDL rigor post-launch.

📌 *8.6 PRSA5: Security Architectural Reviews (Legacy, M&A, EOL)*

- Legacy code: Often undocumented, no ownership, risky.
- M&A: Risk of absorbing insecure or non-compliant codebases.
- End-of-life (EOL): Secure shutdown and documentation.

📈 Key Success Factors:

- Predefined response workflows (e.g., PSIRT).
- Good tooling for tracking issues.
- Organizational buy-in and trained people in all areas (not just devs).

📤 Deliverables:

- External vuln disclosure plan
- Third-party review reports
- Security strategy for legacy, M&A, and EOL
- Post-release certification docs

📊 Metrics:

- Time to respond to external vulns (in hours)
- FTE hours/month spent on disclosure process
- Number of findings post-release
- Number of customer-reported issues
- % of unreported issues found outside SDL

🧠 Chapter Quick-Check Sample Questions:

1. **What is essential for product deployment post-release?**
   ✅ Configuration management
2. **How should incident response processes behave?**
   ✅ Routinely executed and tested
3. **Who makes up the incident management team?**
   ✅ Diverse group from all disciplines
4. **What is sustainment in ops and management?**
   ✅ Ongoing configuration and change management

🧪 Practice Summary:

This chapter is about preparing your **product and org for life after launch**. You're not done at "ship" — you need to actively support your software, hunt for vulns, fix issues, and remain compliant through third-party eyes.

Treat security as a **living process**, not a checkbox.

CHAPTER 9

✅ Chapter 10 Summary: Practical Core Software Security – *Building and Sustaining a Software Security Program*

📌 Chapter Overview

Chapter 10 ties together all the lessons of the book and focuses on building and sustaining a **successful software security program (SSP)**. It explores cultural change, executive buy-in, talent management, and metrics-driven maturity growth.

🎯 *Chapter Take-Aways (With Answers)*

1. **Describe the challenges and opportunities in building a software security program.**

- Challenges: Cultural resistance, lack of executive support, limited resources, poor developer training.
- Opportunities: Strong ROI, brand protection, legal compliance, product quality boost.

2. **Understand the foundational building blocks of a successful software security program.**

- Executive sponsorship
- Clear goals and vision
- Talented cross-functional teams
- SDL integration across SDLC
- Repeatable metrics-driven processes

3. **Define the best practices for gaining executive support and alignment.**

- Present security in **business terms** (risk mitigation, compliance, cost avoidance).
- Provide **real-world breach examples**.
- Align goals with corporate strategy.

4. **Describe the concept of "defense-in-depth" in a software security context.**

- Multiple layers of security (code, network, data, identity, etc.) across the **entire SDLC**.

5. **Explain how to measure, report, and evolve the software security program.**

- Use **maturity models** (like BSIMM, SAMM).
- Track metrics (e.g., vulnerabilities per product, SDL adoption rate, remediation SLAs).
- Iterate improvements via feedback loops.

Key Concepts and Terms

Term	Definition
Software Security Program	A long-term, strategic initiative for securing software across the organization.
Executive Sponsorship	High-level support necessary to fund, promote, and sustain security programs.
Defense-in-Depth	A multi-layered approach to security that protects at various points of failure.
Security Champions	Developers embedded in teams who promote secure coding practices.
Security Metrics	Quantifiable data points like vulnerabilities found/fixed, SDL participation.

Implementation Strategy Steps

6. **Start Small and Scale** – Pilot with one team, learn, adjust, expand.
7. **Identify Security Champions** – Empower influential developers.
8. **Train Continuously** – Make security training role-specific and ongoing.
9. **Incentivize** – Reward good behavior: secure code, fast response, initiative.
10. **Report to Execs Regularly** – Show progress, risks, and ROI.

- Can I identify basic software security concepts and principles used throughout the SDLC?
- Can I determine the importance and types of functional and non-functional software security requirements?
- Can I explain threat and vulnerability taxonomies that are used throughout the software development community?

Software security is essential to protect against malicious attacks so the software can function correctly. This lesson discusses the importance of implementing security early in the software development life cycle and looks at various ways of implementing security to provide integrity, authentication, and availability while doing software development.

Consider the following learning objectives as you move through the lesson:

- Identify basic software security concepts and principles involved throughout the software development life cycle (SDLC).
- Determine applicable secure software coding guidelines and standards.

Practical Core Software Security

- In this digital age, have you ever wondered if your information is safe and secure when you log in to an application or fill out a form on the internet? This lesson describes the importance of software security and application security and looks at the various challenges in the **software development life cycle** (SDLC). The software development life cycle standardizes security best practices. As a result, security has become an essential aspect of every software development and must be considered starting from the design phase. You will look into the quality versus secure code and the implications of neglecting security. As you explore the goals of the **security development life cycle** (SDL), the structured process to enable the production of software, and the role of security in SDLC with application risk analysis, you will gain a better understanding of software security as a whole.

- **Read** chapter 1 "the book" from *Practical Core Software Security: A Reference Framework* for a more in-depth understanding of software security during software development.

Software and Systems Security for CompTIA CySA+

As you consider security within your organization, you need to understand both software and hardware securities. Software refers to any programs or operating systems that your organization uses. Hardware describes the physical components of a computer. In this lesson, you will learn to evaluate and integrate security in software and hardware and discover the different phases and models of SDLC while also considering their advantages and disadvantages. This will help determine what is best for the given organization. The need for code reviews is crucial in deciding security systems, so the second part of the lesson looks at best practices for coding securely. Finally, the lesson discusses SOA, microservices architectures, and infrastructure-as-code in cloud computing.

Watch Software and Systems Security for CompTIA CySA+ (02:23:00) to learn more about how to evaluate and integrate security within both software and hardware used by your organization.

References

Meredith, D. (2020, Oct 7). *Software and systems security for CompTIA CySA+* [Video]. PluralSight. https://app.pluralsight.com/library/courses/software-systems-security-comptia-cysa-plus/table-of-contents

Ransome, J., Misra, A. and Merkow, M. (2023). *Practical core software security: A reference framework*. CRC Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=3298314&site=eds-live&scope=site&authtype=sso&custid=ns017578&ebv=EB&ppid=pp_1

- It's crucial to identify the importance, relevance, and cost of building security into your software.
- SDLC stands for the software design life cycle.
- SDL stands for security development life cycle.
- Software security entails building security into the software through an SDL in an SDLC.
- The three core elements of security are confidentiality, integrity, and availability.
- Threat modeling and attack surface validation throughout the SDL will alleviate security vulnerabilities.
- Integrating and evaluating software and hardware used by your organization will maximize your organization's software and security.
- There are eight major phases of the SDLC: planning, requirements, design, implementation, testing, deployment, maintenance, and end of life.

Key Terms

- **hardware:** the physical components of a computer or electronic system
- **deployment phase:** during this phase of the SDLC, security is pushed out
- **design phase:** during this phase of the SDLC, requirements are prepared for the technical design
- **end of life phase:** during this phase of the SDLC, the proper steps for removing software completely are considered
- **implementation phase:** during this phase of the SDLC, the resources involved in the application from a known resource are determined
- **maintenance phase:** during this phase of the SDLC, ongoing security monitoring is implemented
- **planning phase:** during this phase of the SDLC, a vision and next steps are created
- **requirement phase:** during this phase of the SDLC, necessary software requirements are determined
- **secure code**: a principle design in coding that refers to code security best practices, safeguards, and protection against vulnerabilities
- **security development life cycle (SDL)**: a process that standardizes security best practices
- **software**: the program and operating systems used by a computer
- **software development life cycle (SDLC)**: a structured process that enables the production of software
- **testing phase:** during this phase of the SDLC, software is tested to verify its functions through a known environment
- **threat modeling**: a structured process to protect against vulnerabilities

Describe security standards, best practices, and tools to achieve secure development life cycle (DL) and software development life cycle (SDLC) goals.

The Security Development Life Cycle

The security development life cycle standardized security best practices, but does that mean every organization should have the same security? Of course not! This lesson will introduce different security models which may be appropriate for multiple organizations. First, you will start by discussing the challenges in securing applications with examples of different types of attacks. Later on, you will dive deeper into the two security models: **Building Security in Maturing Model (BSIMM)** and **Open Web Application Security Project (OWASP)**. BSIMM studies real-world software security, while OWASP is a flexible and prescriptive framework that helps build security into your software development organization. These models put forth studies and perspectives to improve software security in our ever-changing world. Additionally, you will read about various resources available for SDL best practices, including guidelines from the **National Institute of Standards and Technology (NIST)** and the US government. The NIST provides research, information, and tools for both government and corporate information security. You will also explore the various tools and techniques available for code review and gathering meaningful security metrics; the lesson ends with mapping SDL to SDLC phases and discussing the different SDLC methodologies.

Read chapter 2 "The Security Development Lifecycle" from *Practical Core Software Security: A Reference Framework* to continue to learn about the security development life cycle and the challenges it may face.

Evolution of Application Security

In this section of the lesson, you will watch a video regarding the evolution of **application security**. Application security describes the concept that the online world is constantly evolving. To keep up, it is imperative to develop, add, and test security features continuously. This video considers how modern-day application security programs achieve success. Garrett Gross and Alyssa Miller discuss the open-source community, which has grown exponentially and includes many libraries and packages available to use. They also discuss the DevOps pipeline, the various tools available in the market, and how you can choose tools specific to your use case. Software security professionals should feel empowered to integrate security with a developer-centric mindset; this allows development teams to trust that all team members are aligned in their mission to release good, secure software.

Watch Shift Left, Shift Right: The DevSecOps Hokey Pokey (00:30:36) to gain a deeper insight into the evolution of application security.

Check out NIST's AI framework. NIST has developed numerous standards for Computer Security, so explore their influence on AI.

AI Risk Management Framework

References

Gross, G. and Miller, A. (2020, Oct 6). *Shift left, shift right: The DevSecOps hokey pokey* [Video]. PluralSight. https://app.pluralsight.com/library/courses/allthetalks-session-52/table-of-contents

Ransome, J., Misra, A. and Merkow, M. (2023). *Practical core software security: A reference framework*. CRC Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=3298314&site=eds-live&scope=site&authtype=sso&custid=ns017578&ebv=EB&ppid=pp_1

Take a moment to think about what you learned in this lesson.

- Implementing an SDL program ensures that security is built into a software design rather than an afterthought.
- Some popular SDL models are BSIMM SSDL Touchpoints, the OWASP Code Review Guide, the Cisco Secure Development Life Cycle (Cisco SDL), and Microsoft's Trustworthy Computing Security Development Life Cycle.
- Building Security In Maturing Model (BSIMM) studies real-world software security initiatives. It allows you to determine where your software security stands and how to develop it over time.
- There are 12 best BSIMM practices.
- The OWASP Software Assurance Maturity Model (SAMM) is a flexible and prescriptive framework for building security into software development organizations.
- The National Institute of Standards and Technology (NIST) provides research, information, and tools for government and corporate information security.
- The US Department of Homeland Security has an established Software Assurance Program.
- Common Computer Vulnerabilities and Exposures (CVE) is a list of information that aims to provide common names for publicly known security vulnerabilities.
- You must map whatever form of SDL you use to your current SDLC.

- Security metrics allow for corporations to make decisions regarding risk management requirements and security budgets and to show customers proof of security.
- Application security is the process of developing, adding, and testing security features within applications.
- Application security aims to prevent security vulnerabilities against threats.

Key Terms

- **application security:** developing, adding, and testing security features to prevent vulnerabilities within applications
- **Building Security In Maturing Model (BSIMM):** a study of real-world software security that allows you to develop your software security over time
- **dynamic analysis:** the analysis of computer software that is performed when executing programs on a real or virtual processor in real time
- **fuzz testing:** automated or semi-automated testing that provides invalid, unexpected, or random data to the computer software program
- **National Institute of Standards and Technology (NIST):** provides research, information, and tools for government and corporate information security
- **measurement model:** a set of data security methods that developers take to protect against vulnerabilities
- **metric model:** allows an organization to determine the effectiveness of its security controls
- **Open Web Application Security Project (OWASP):** a flexible and prospective framework to build security into your software development organization
- **static analysis:** the analysis of computer software that is performed without executing programs

Explain the advantages, disadvantages, and software development roles within the Agile, Waterfall, and DevSecOps frameworks.

Software Development Methodologies

You may already be familiar with the terms **Agile**, **Waterfall**, and maybe even the **V-Model**. However, have you considered the impact of their histories on the way they are used today? In this lesson, you will learn about these methodologies in greater detail. The Waterfall methodology is a sequential, step-by-step process for addressing an organization's requirements. The V-Model follows Waterfall methodology but then turns back upwards after the coding process. The Agile methodology mixes traditional and new software development practices. You will dive deep into their history, how they work, and their advantages and disadvantages. You will also explore how development teams might use **Scrum** – a newly popular, flexible, and holistic product development strategy – to work as a unit to reach a common goal. After learning the ins and outs of these methodologies, you will be able to use this information to make informed decisions about how to apply them in your organizations moving forward.

Watch Agile Fundamentals (02:40:00) to learn more.

Agile vs. Waterfall

Now it's time to go into even greater detail about the Agile and Waterfall models, their differences, and approaches to choosing a model. Scrum, its framework, and its methodology will also be discussed, as well as the various Scrum roles and values. After watching this video, you should feel more confident choosing the best methodology for your organization.

Watch Learning the Truth about Agile vs. Waterfall (01:05:00) to explore more about the Agile and Waterfall models.

References

Cobb, C. (2020, May 7). *Learning the truth about Agile versus Waterfall* [Video]. PluralSight. https://app.pluralsight.com/library/courses/learning-agile-versus-waterfall/table-of-contents

Haunts, S. (2020, Jan 1). *Agile fundamentals* [Video]. PluralSight. https://app.pluralsight.com/library/courses/agile-fundamentals/table-of-contents

ake a moment to think about what you learned in this lesson.

- The Waterfall approach divides the process of software development into separate phases. The outcome of one phase acts as the input for the next phase.
- An advantage of the Waterfall method is splitting project deliveries into different stages, making it easier for an organization to control the development process.
- A disadvantage of the Waterfall method is that it does not allow time for reflection or revision to a design.
- A V-model is not designed to be linear, but instead, the stage is turned back upward after the coding phase is complete, creating the V shape.
- The Agile methodology mixes traditional and new software development practices.
- Agile software development uses collaboration between self-organizing and cross-functional teams.
- Agile has four core values and 12 principles that can be followed.
- Agile framework allows for customer satisfaction through rapid, continuous delivery of useful software.
- A disadvantage of Agile is that it is difficult to assess the effort required at the beginning of the SDL.
- Scrum framework allows for a development team to work flexibly and holistically to reach a common goal.
- Extreme programming intends to improve software quality and responsiveness.
- Extreme programming is a type of Agile software development.
- Consider your software development projects and determine which approach is most suitable for you.

Key Terms

- **Agile methodology:** mixes traditional and new software development practices
- **extreme programming (XP):** a software development methodology that is intended to improve software quality and responsiveness
- **Scrum:** flexible, holistic product development strategy where a development team works as a unit to reach a common goal
- **V-model:** a variation of the waterfall model, where the stage is turned back upwards after the coding phase
- **Waterfall methodology:** a sequential, step-by-step process for requirements

This section, Software Requirement and Risks, covers the first two phases of the security development life cycle (SDL): the Security Assessment phase and the Architecture phase. In some SDLs, the Security Assessment phase is known as the Discovery phase.

During the Security Assessment phase, the project team identifies the risk profile and needed SDL activities of the software product. An initial project outline for security is developed and merged into the overall project development schedule to allow project members to respond as changes occur. The security assessment should determine how critical the product is to meeting customer needs, what security objectives are required, what regulations and policies are applicable, and what threats are possible in the product's operating environment. The assessment team should be included in project kickoff meetings to ensure discussions about security are happening as early as possible.

Software security teams need to define how security should work within the product and to verify the functionality exists. To do that, they interview product stakeholders during the security assessment to determine security requirements. Those security requirements are added to the product backlog, just like functional application requirements.

This section covers one competency in two lessons:

- **Lesson 4: Security Assessment Planning**
- **Lesson 5: Architecture Security Analysis**

Build Employability Skills and Competence

Software security has come under increased scrutiny over the last 10–15 years due to a number of high-profile cases where private information has been stolen. Since security breaches can cost companies millions of dollars, opportunities in cybersecurity and software security have multiplied in recent years. IT employees that aren't directly in the security space are expected to have foundational knowledge of software security threats.

The skills in this section are:

1. Define the requirements for a software solution.
2. Determine which security measures are required based on client requirements.
3. Ensure compliance with organizational information privacy.

In this section, you can demonstrate these skills by passing a final assessment that measures whether you have mastered this competency:

- The learner assesses software requirements and risks to ensure threats are addressed.

Prepare for the Assessment

To prepare for the objective assessment, ask yourself these questions:

- **Can I identify the key components of the Security Assessment (A1) phase?**
- **Can I explain threat modeling methodology and how an organization should go about choosing one?**

1. Identify requirements, tools, and techniques for software solutions that ensure security threats are addressed and mitigated.

🔐 **Security Requirements**

- **Authentication & Authorization** (e.g., role-based access, MFA)
- **Data Protection** (e.g., encryption in transit and at rest)
- **Input Validation** (prevent injection attacks)
- **Error Handling** (no information leakage)
- **Audit Logging** (forensics, compliance)

🛠️ **Tools**

- **Static Application Security Testing (SAST)** – e.g., SonarQube, Fortify
- **Dynamic Application Security Testing (DAST)** – e.g., OWASP ZAP, Burp Suite
- **Software Composition Analysis (SCA)** – e.g., Black Duck, Snyk
- **Threat Modeling Tools** – e.g., Microsoft Threat Modeling Tool
- **Security Linters** – e.g., Bandit (Python), ESLint plugins for JS security

⚙️ **Techniques**

- **Threat Modeling (STRIDE, PASTA)**
- **Secure Coding Standards** (e.g., OWASP Top 10, SEI CERT)
- **Penetration Testing**
- **Automated Security Scanning in CI/CD Pipelines**
- **Defense-in-Depth Architecture** (layered controls)

---

2. Identify privacy requirements that inform software solutions to meet business needs.

🔒 **Common Privacy Requirements**

- **Data Minimization** – collect only necessary data
- **Purpose Limitation** – use data only for intended business processes

- **Consent Management** – collect and track user consents (GDPR, CCPA)
- **Right to Access/Delete** – user data subject rights under privacy laws
- **Data Retention Policies** – automatic expiration of stored PII
- **Anonymization/Pseudonymization** – when full data is not needed

📋 Relevant Regulations

- **GDPR** – Europe (rights of individuals, breach notification)
- **CCPA** – California Consumer Privacy Act
- **HIPAA** – U.S. healthcare
- **FERPA, PCI-DSS, etc.**

🧩 How They Inform Software Design

- Influence **data models** (e.g., tagging PII fields)
- Require **privacy impact assessments** (PIAs)
- May force specific **encryption standards**, **logging redactions**, etc.

---

3. Identify potential constraints on functionality and systems integrations from security controls.

🛑 Potential Constraints

- **Performance Impacts** – due to encryption, logging, scanning
- **Delayed Deployment** – extra testing or remediation cycles
- **Third-Party Integration Issues** – partners may not meet your security standard
- **Legacy System Compatibility** – may lack support for modern security controls
- **User Friction** – MFA, CAPTCHA, or consent prompts may reduce usability
- **Data Sharing Restrictions** – privacy laws may limit API or analytics integrations

📉 Examples

- SSO implementation might require refactoring of auth logic
- Data masking may reduce analytics accuracy
- Rate-limiting for DDoS protection might disrupt bulk data transfers

Security Assessment

A software security assessment is an important part of the overall software development life cycle. The assessment should be performed in the earliest phases of any new production development initiative and should identify the following:

- How critical is the product to meeting the needs of customers?
- What security objectives are required by the product?
- Are there any regulations or policies that govern the product data?
- What threats are possible once the product has been deployed to customers?

Read chapter 3 "Security Assessment (A1): SDL Activities and Best Practices" from *Practical Core Software Security: A Reference Framework* to better understand the activities performed in a security assessment and why they are so important.

Secure Software Requirements for CSSLP

As with most organizations, there are multiple levels of development on the course to a final project. Software security and development is no different. Software is written by developers according to requirements defined by analysts and based on interviews with customers and product owners; those requirements define business functions so that technical staff can propose software solutions. Secure software requirements focus on confidentiality, integrity, and availability of information within the product and are a subset of the functional business requirements.

Watch the section "Discovering Secure Software Requirements" (00:30:53) from *Secure Software Requirements for CSSLP* to learn more about gathering software requirements.

References

Henry, K. (2023, July 7). *Secure software requirements for CSSLP®* [Video]. PluralSight. https://app.pluralsight.com/library/courses/secure-software-requirements-csslp-cert/table-of-contents

Ransome, J., Misra, A. and Merkow, M. (2023). *Practical core software security: A reference framework*. CRC Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=3298314&site=eds-live&scope=site&authtype=sso&custid=ns017578&ebv=EB&ppid=pp_1

Take a moment to think about what you learned in this lesson.

- The Security Assessment (A1) phase is the first phase of the security development life cycle.
- During the Security Assessment (A1) phase, an initial project outline for security milestones is developed and integrated into the development project schedule.
- During the Security Assessment (A1) phase, all key stakeholders should discuss, identify, and have a common understanding of the security and privacy implications, considerations, and requirements.
- The software security team should be included in the software development life cycles (SDLCs) kick-off meetings to ensure that security is a key element of the SDLC and built into the process.
- A privacy impact assessment should include the summary of the legislation, required process steps, technologies and techniques, and any additional resources.
- Creating success criteria for any SDL phase makes it more effective and, in postmortem, helps identify what worked and what didn't.
- Creating a key set of deliverables for each SDL phase allows for all required activities to have tangible documented outcomes.
- In the SDL model, it is helpful to outline metrics that should be measured in each and every phase.
- The three areas of focus in secure software requirements are gathering the software requirements, data classification, and managing data protection requirements.
- The purpose of gathering the software requirements before a project kick-off is to avoid common project failures by identifying requirements at the beginning of a project.
- When identifying functional and non-functional requirements, consider the current organization as well as future business needs.
- Operational requirements refer to how the system should function based on the environment in which the system will operate.
- The three areas of compliance requirements are legal, financial, and industry standards.

Key Terms

- **functional requirements**: requirements that describe what the system will do and its core purpose
- **non-functional requirements**: requirements that describe any constraints or restrictions on a design but do not impact the core purpose of the system
- **privacy impact assessment**: a process that evaluates issues and privacy impact rating in relation to the privacy of personally identifiable information in the software
- **product risk profile**: helps to determine the actual cost of the product from different perspectives
- **requirement traceability matrix**: a table that lists all of the security requirements
- **Security Assessment (A1) phase**: the first phase of the security development life cycle in which the project team identifies the product risks and creates a project outline for security milestones
- **threat profile**: the environment in which the product will operate and potential threats in that environment

In this lesson, you will learn about the Architecture (A2) phase. During the Architecture phase, the project team brings security considerations into the software development life cycle (SDLC) to ensure threats, requirements, and potential pain points are addressed. The initial security assessment has been completed and direction has been set. In this phase, teams assess and scope security policies, perform threat modeling, and plan countermeasures to attacks.

Consider the following learning objectives as you move through the lesson:

- Identify organizational policies to comply with when designing software solutions.
- Identify actions taken during each step in threat modeling for the analysis of architecture security.

Architecture (A2)

The **Architecture (A2) phase** examines security in terms of business risks, with inputs from the software security team and key stakeholders. During the Architecture phase (A2), software security teams will review any policy not governed by internal SDL policies and instruct product teams on what features and policies to implement. They will also assess and scope internal SDL policies without disrupting product development teams. The most important step in this phase, however, is **threat modeling**. In threat modeling, technicians identify security objectives, survey applications, decompose applications, identify threats, and identify vulnerabilities.

Read chapter 4 "Architecture (A2): SDL Activities and Best Practices" from *Practical Core Software Security: A Reference Framework* to better understand the activities performed in a security assessment and their importance.

Threat Models Fundamentals

Organizations have cracks, and hackers know how to identify these vulnerabilities to infiltrate an organization. Threat modeling requires a technician to think like an adversary. The goal is to understand a threat and how to defend against it before it happens. Threat modeling involves the use of a repeatable model to identify vulnerabilities so product developers can eliminate them.

Watch Threat Modeling Fundamentals (01:29:00) to explore more about threat modeling.

Finding Threats Using STRIDE

Threat modeling is performed by diagramming a process and then identifying possible threats to each element of the process. Technicians use data flow diagrams to create a visual representation of all of the interactions with each element in the process. The software security team then examines the data flow diagram to identify vulnerabilities. STRIDE is an acronym in which each letter represents a type of threat to software. Using the methodology allows the team to investigate each element in their data flow diagram for the vulnerabilities represented by each letter of the acronym. The team may perform STRIDE-per-element, which identifies threats for interactions with elements, or they may perform STRIDE-per-interaction, which identifies threats for interactions between elements.

Watch the section "Finding Threats Using STRIDE" (00:19:38) from *Performing Threat Modeling with the Microsoft Threats Modeling Methodology* in order to understand the STRIDE methodology more.

PASTA Methodology

Another threat modeling methodology/acronym is PASTA. PASTA, or "process for attack simulation and threat analysis," is a seven-step methodology that, again, gives a software security team a repeatable framework for identifying threats and planning countermeasures. Unlike other methodologies that focus on the identification of threats, PASTA focuses on the probability of those threats and their corresponding impact. It ensures that teams focus on threats that represent a real impact on their products.

Watch Perform Threat Modeling with the PASTA Methodology (01:04:00) to dive deeper into the PASTA methodology.

References

Boyer, J. (2018, Oct 1). "Performing threats using STRIDE." *Performing threat modeling with the Microsoft threat modeling methodology* [Video]. PluralSight. https://app.pluralsight.com/course-player?clipId=a148b426-9b42-448b-8146-c5d1338da3e9

Mosmans, P. (2017, Aug 13). *Threat modeling fundamentals* [Video]. PluralSight. https://app.pluralsight.com/library/courses/threat-modeling-fundamentals/table-of-contents

Pandey, P. (2020, Oct 19). *Performing threat modeling with the PASTA methodology* [Video]. PluralSight https://app.pluralsight.com/library/courses/performing-threat-modeling-pasta/table-of-contents

Ransome, J., Misra, A. and Merkow, M. (2023). *Practical core software security: A reference framework*. CRC Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=3298314&site=eds-live&scope=site&authtype=sso&custid=ns017578&ebv=EB&ppid=pp_1

- The second phase of the security development life cycle involves bringing security considerations into the software development life cycle.
- The software security policy defines what needs to be protected and how it will be protected.
- Collaboration among the privacy function, the centralized group, or outside legal counsel during software security design represents many organizations' best practice.

- Threat modeling is a process to pinpoint security threats and potential vulnerabilities that will help prioritize remediation.
- Threat modeling proactively prepares an organization for potential threats rather than reacting after threats are discovered.
- Consider the business or organization to help determine the best approach for threat modeling.
- The following five steps of threat modeling will help an organization better understand how to best protect its assets: identify security objectives, survey the application, decompose it, identify threats, and identify vulnerabilities.
- Data flow diagrams provide a visual representation of a process flow.
- Threats can be categorized by type: spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege, which make up the acronym STRIDE.
- PASTA methodology of threat modeling stands for the process of attack simulation and threat analysis.
- After identifying threats, it is important to consider the design and implementation aspects of your software application to rank threats and vulnerabilities to your organization.
- Rank the organization's threats based on their probability and damage potential.
- The DREAD model is one of the most popular risk models. The DREAD model consists of damage potential, reproducibility, exploitability, affected users, and discoverability.
- The Web Application Security Frame (aka Application Security Frame) uses categories to organize common security vulnerabilities with a focus on web software applications.
- Trike is a framework for security auditing from a risk management perspective.

Key Terms

- **application-centric threat modeling**: threat models that start with visualizing the application you are building
- **asset-centric threat modeling**: threat models focused around senior management and protecting the assets of an organization
- **application decomposition**: determining the fundamental functions of an app
- **Architecture (A2) phase**: the second phase of the security development life cycle that examines security from perspective of business risks
- **data flow diagrams**: a visual representation of the threat flow
- **denial of service**: denying access to valid users
- **elevation of privilege**: privileged access to resources for gaining unauthorized access to information
- **information disclosure**: read a file that one was not granted access too
- **PASTA**: the process for attack simulation and threat analysis that gives a software security team a repeatable framework for identifying threats
- **repudiation**: performing illegal operations in a system that lacks the ability to trace the prohibited operations
- **risk model**: assess vulnerabilities during the software development process
- **software security policy**: defines what needs to be protected and how it will be protected
- **spoofing**: illegally accessing and using another user's credentials
- **tampering**: maliciously changing or modifying persistent data
- **third party codes**: reusable software developed externally from the organization's platforms
- **threat modeling**: a process to pinpoint security threats and potential vulnerabilities that will help prioritize remediation
- **threat source**: the entity carrying out the attack
- **threat vector**: the path an attacker can take to exploit a vulnerability
- **Trike**: a unified conceptual framework for security auditing
- **vulnerability**: a weakness that can be exploited

Section 2 introduced the first two phases of the Security Development Life Cycle (SDL). The security assessment is where the project team identifies the product risk profile and what SDL activities are needed.

The initial project outline for security milestones and controls is developed and integrated into the overall project schedule to allow proper planning as changes occur. The Architecture phase defines business requirements in terms of confidentiality, integrity, and availability and also identifies privacy controls that must be implemented to prevent leaking personally identifiable information (PII).

Security teams perform threat modeling in the Architecture phase by creating data flow diagrams of business processes and identifying threats to each element of the product design.

DREAD = **Damage**, **Reproducibility**, **Exploitability**, **Affected users**, **Discoverability**

Element	Symbol	Explanation
Trust Boundary	**Dashed line**	Separates zones of differing trust (e.g., user → app)
Data Flow	**Solid line with an arrow**	Shows direction of data movement
Data Store	**Two parallel horizontal lines**	Represents a storage mechanism (DB, file system)
External Element	**Rectangle**	An entity outside the system's control (e.g., user, 3rd-party API)
Process	**Circle or rounded rectangle**	Represents logic, computation, or transformation

Section 3: Introduction

Have you wondered how organizations build and release software into the market, especially organizations that handle personal identifiable information (PII) data like insurance, healthcare, banks, and finance companies?

There is a lot that goes into the design and development of these applications handling PII data, especially regarding compliance and privacy rules. This section, Software Security Test Plan, evaluates the components of a security test plan and identifies methods to automate security testing. It provides an overview of the software security analysis tools, including static analysis, dynamic analysis, fuzz testing, and code review. Based on the outcome of the security test plan, you will learn more about the guidance on adjusting security controls for developing secure software.

This section covers one competency in two lessons

- Lesson 6: Security Test Planning
- Lesson 7: Security Test Case Execution

Build Employability Skills and Competence

This section will prepare you for careers within the software security test planning phase. As you work through Section 3 to learn more about software security analysis tools, you will gain an understanding of how to design and implement technical solutions, which will help you as a developer to identify security weaknesses. You will gain knowledge and experience in using popular application security tools such as SonarQube, OWASP Zed Attack Proxy, and Snyk. As you build your confidence in the software security world, you will be able to provide guidance, technical leadership, and direction to an application security team. The skills you gain from this section will allow you to conduct reviews of security posture, which will ensure that application systems are being operated securely. These dynamics will reach many people as we live in an ever-changing, technology-driven society.

The skills covered in this section are:

1. Evaluate methods for testing the effectiveness and efficiency of information security controls.
2. Implement static application security testing.
3. Describe information security controls.

In this section, you can demonstrate these skills by passing a final assessment that measures whether you have mastered this competency:

- The learner evaluates software security test plan documentation and implementation strategies.

Prepare for the Assessment

To prepare for the objective assessment, ask yourself these questions:

- Can I create and evaluate a security test plan?
- Can I determine the best software security analysis tool for my application?
- Can I make proper adjustments to the security of my software based on security test plan results?

✅ **1. Can I create and evaluate a security test plan?**

Yes, if you can:

- **Identify security objectives for the application (e.g., protect data, enforce roles)**
- **Select test types:**
    - **Static analysis (SAST)**
    - **Dynamic analysis (DAST)**
    - **Manual penetration testing**
    - **Fuzz testing**
- **Define test coverage:**
    - **Input validation**
    - **Authentication & access control**
    - **Session management**
    - **Error handling**
- **Include pass/fail criteria**
- **Prioritize tests by risk level**
- **Evaluate the plan by comparing:**
    - **Test results vs. security requirements**

- **Whether critical vulnerabilities are covered**

📌 **Tip: Reference the SDL A4 phase — *Security Test Planning* and *Secure Code Review* deliverables are key here.**

✅ **2. Can I determine the best software security analysis tool for my application?**

Yes, by knowing:

- **SAST (Static Application Security Testing):**
  - Use before runtime
  - Finds source code flaws
  - Tools: SonarQube, Fortify, Checkmarx
- **DAST (Dynamic Application Security Testing):**
  - Run while app is executing
  - Finds runtime issues like XSS, SQLi
  - Tools: OWASP ZAP, Burp Suite
- **SCA (Software Composition Analysis):**
  - Finds third-party dependency risks
  - Tools: Snyk, Black Duck
- **Fuzzing Tools:**
  - Send random/malformed inputs to crash app
  - Tools: Peach Fuzzer, AFL

📌 **Match the tool type to the stage in development and risk profile of your product.**

✅ 3. Can I make proper adjustments to the security of my software based on security test plan results?

Yes, if you:

- **Review test output critically:**
    - Prioritize high-severity vulnerabilities (e.g., RCE, injection)
    - Fix issues based on threat modeling results
- **Apply secure coding techniques:**
    - Input validation (e.g., whitelist)
    - Use prepared statements to prevent SQLi
    - Set secure session timeouts
- Re-test after applying fixes
- Update documentation and risk registers
- Loop findings into continuous integration pipeline

📌 Use the test results to guide code remediation, re-testing, and post-release patch planning (ties into SDL PRSA1–PRSA5).

✅ 1. Evaluate the components of a security test plan

A **Security Test Plan** defines how to validate that software meets its **security objectives**. It must cover:

🔍 Key Components:

Component	Description
Test objectives	What security goals are being validated (e.g., auth, access control, data)?
Scope	What modules/functions are covered (APIs, DBs, login systems)?
Test types & tools	Which types of testing: - **SAST**: code-level static analysis - **DAST**: black-box runtime testing

Component	Description
	- **Fuzzing**: random input testing - **Manual pen tests**: real-world exploitation
Roles & responsibilities	Who executes the tests, who fixes bugs
Pass/fail criteria	When is a test considered successful? Severity thresholds?
Schedule & frequency	When and how often testing will occur (e.g., CI/CD pipeline)?
Remediation steps	How vulnerabilities are prioritized and addressed
Re-test and verification	Retesting to ensure issues are resolved and haven't regressed

📌 **Related SDL Phases**: A4 (Security Testing), A3 (Secure Code Review)

✅ 2. Identify methods to ensure defense in depth software controls

Defense in Depth = **Layered security controls** to reduce risk if one layer is bypassed.

🛡️ Key Methods:

Layer	Example Defense Controls
Input layer (User → App)	- Input validation, encoding, CAPTCHA
Authentication	- Multi-factor authentication (MFA)
Authorization	- Role-based access control (RBAC), least privilege
Data protection	- Encryption at rest/in transit (TLS, AES), tokenization
Application logic	- Secure session handling, no hardcoded secrets
Codebase	- Static analysis, secure libraries only
Network	- Firewalls, API gateways, segmentation
Monitoring & Logging	- Audit trails, SIEM alerts, tamper-resistant logs
Post-deployment	- Patch management, vulnerability scanning, bug bounty programs

🔁 SDL Tie-In:

- Applied across all phases: from **threat modeling (A2)** to **code review (A3)** to **patching (PRSA)**.
- **Defense-in-depth** ensures no single point of failure exists.

Design and Development (A3)

Functional and design specifications help to establish best practices during the **Design and Development (A3)** phase of the SDL. During the Design and Development phase, you complete analysis and tests in order to make informed decisions about how to deploy your software security. This is when you will establish best practices to detect and remove security and privacy issues. To do this successfully, look at a compliance analysis, including policies from outside organizations that set security and privacy requirements.

This lesson discusses the various security testing methods and how to prepare a security test plan to validate the secure implementation of a product. It also reviews updates to the threat model, privacy assessment, and completion of a security analysis and review.

Read chapter 5 "Design and Development (A3): SDL Activities and Best Practices" from *Practical Core Software Security: A Reference Framework*.

Secure Software Considerations

Developing security controls is an essential component in secure software design. This video shows the considerations that go into developing controls in a software project.

Watch "Secure Software Considerations" (00:44:31) from *Secure Software Architecture and Design for CSSLP* to learn more about controls.

Security Test Cases

There are many types of security test cases to explore. These include:

- **Testing the box**
- **Vulnerability assessment and penetration testing**
- **Scanning**
- **Ongoing testing**
- **Environment testing**

This lesson explains how to develop security test cases and build a testing strategy for systems-related and non-systems-related functions. Additionally, it details the execution of a test plan, including the importance of documentation, verification, and validation.

Watch the section "Develop Security Test Cases" (00:58:02) from *CSSLP Software Security Testing* to learn more. Please note: You will watch the rest of this video later on in this section.

References

Henry, K. (2020, Nov 19). *Secure software testing for CSSLP®* [Video]. PluralSight https://app.pluralsight.com/course-player?clipId=eca4bc1d-0432-4b71-8ee6-dc871c5d0ad3

Henry, K. (2023, Aug 9). *Secure software architecture and design for CSSLP®* [Video]. PluralSight. https://app.pluralsight.com/library/courses/secure-software-architecture-design-csslp-cert/table-of-contents

Ransome, J., Misra, A. and Merkow, M. (2023). *Practical core software security: A reference framework*. CRC Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=3298314&site=eds-live&scope=site&authtype=sso&custid=ns017578&ebv=EB&ppid=pp_1

Take a moment to think about what you learned in this lesson.

- During the A3 phase, any policy that exists outside of the SDL policy is reviewed.
- Both the software security group and the centralized information security group must collaborate on all policies and guidelines.
- The purpose of testing activities is to validate the security of software before making the product available.
- The goal is to build security in, which is less costly than correcting problems discovered after the software has been deployed.
- The test environment should mimic the execution environment as closely as possible.
- Software security testing techniques are categorized by white box, gray box, or black box testing.
- Developers test their systems to ensure that their code is working properly; this is known as alpha testing.
- External testing, known as beta testing, allows you to check usability and vulnerabilities in a system.
- Security test cases allow a software developer to determine security issues at the lowest level.
- The process of scanning involves identifying deficiencies anywhere around the system.
- Security testing is not static; it is ongoing.
- Applications need to be tested not only in the lab or development area but also in their true operational environment.

Key Terms

- **alpha level testing**: testing done by the developers themselves
- **beta level testing**: testing done by those not familiar with the actual development of the system
- **black box testing**: tests from an external perspective with no prior knowledge of the software
- **Design and Development (A3) phase**: the third phase of the security development life cycle, in which you analyze and test software to determine security and privacy issues as you make informed decisions moving forward with your software
- **external resources**: resources hired on a temporary basis to come into a project, test the application, and report findings
- **functional testing scripts**: step-by-step instructions for a specific scenario or situation
- **gray box testing**: analyzes the source code for the software to help design the test cases
- **internal resources**: resources from the company's organization
- **secure testing scripts**: scripts created specifically for the application being tested
- **scripts**: detailed, logical steps of instructions to tell a person or tool what to do during the testing
- **system test**: test the system and its interaction with other systems
- **white box testing**: tests from an internal perspective with full knowledge of the software
- **vulnerability assessments**: examining a product to identify security deficiencies

✅ D487 BOOK-ALIGNED: Evaluating Software Security Analysis Tools

1. Static Analysis Tools (SAST)

- Analyze **source or binary code** *without executing* it.
- Used **early in the SDL** (Design/Implementation phase).
- Detects syntax issues, insecure coding patterns, and known flaws.
- Tools: Fortify SCA, SonarQube, Checkmarx.

🧠 *Best for identifying issues before code runs. Can be automated in CI/CD.*

2. Dynamic Analysis Tools (DAST)

- Analyze **running applications**.
- Simulate real-world attacks during execution.
- Detects runtime issues like **authentication failures, session hijacking, injection attacks**.
- Tools: OWASP ZAP, Burp Suite, IBM AppScan.

🧠 *Use during testing phases (after build). Complements static analysis.*

3. Fuzz Testing

- Sends **unexpected, malformed, or random inputs** to app interfaces.
- Finds unknown, zero-day vulnerabilities, especially in **input handling**.
- Good for: **Crash resistance, memory corruption, buffer overflows**.
- Tools: AFL, Peach Fuzzer.

🧠 *Use for complex or legacy systems where source code isn't easily auditable.*

4. Code Review (Manual/Peer Review)

- Involves **human inspection** of code.
- Identifies **logic flaws, architecture issues**, security assumptions.
- Formal or informal. Can be checklist-based or tool-assisted.

🧠 *Catches what tools may miss. Encourages developer accountability.*

✅ Book-Aligned: Adjusting Security Controls Based on Test Plan Results

Let's say the **Security Test Plan** from SDL testing reveals issues. Here's how you'd **adjust security controls**:

Test Plan Finding	Recommended Control Adjustment	SDL Phase
Unvalidated input leads to XSS	Add input validation and output encoding filters	Development
Weak password policy discovered via dynamic analysis	Enforce strong password complexity + rate limiting	Architecture/Design
Static tool shows hardcoded secrets in source code	Migrate secrets to secure storage/vaults (e.g., HashiCorp Vault)	Implementation
Fuzzing causes app crash	Improve exception handling and error sanitization	Development
Manual review reveals flawed role-based access control	Redesign RBAC logic; apply principle of least privilege	Architecture/Design
DAST detects unpatched vulnerable component	Update dependencies and re-run software composition analysis (SCA)	Verification/Testing

---

✅ Summary (Straight from SDL/Book Context)

- **Static Analysis** = early detection, fast feedback, code-level issues.
- **Dynamic Analysis** = runtime behavior, auth/session/input bugs.
- **Fuzzing** = random chaos testing for unexpected edge cases.
- **Code Review** = logical flaws and best practice enforcement.
- **Adjustments** must be mapped to SDL phases and follow defense-in-depth.

Design and Development (A4)

As you move into Phase 4 of the SDL, you begin to gather information regarding **Design and Development (A4)**. This is a continuation of Phase 3, where the elements of security test case execution are described. This lesson outlines how to document the key success factors for completion of Phase A4 and identifies why it is important to include security testing as part of quality assurance (QA) testing so the security team can focus on advanced threats cases. It also describes the various code-review techniques and how they can be very cost-effective in locating and fixing security vulnerabilities in the early stages of development.

Read chapter 6 "Design and Development (A4): SDL Activities and Best Practices" from *Practical Core Software Security: A Reference Framework*.

Modern Dynamic Application Security Testing

AppSec is the overall process of identifying, fixing, and preventing security vulnerabilities within the application level, which is a crucial part of the software development life cycle. In this video, Scott Gerlach – a CSO and co-founder of StackHawk who was previously the senior security architect at GoDaddy – introduces you to the issues with AppSec and how to use static code analysis and dynamic code analysis. This helps developers get a head start on their Continuous Integration/Continuous Delivery (CI/CD) pipeline by introducing these tools early on in the software development life cycle.

Watch Modern Dynamic Application Security Testing (00:13:00) by DevSecCon.

Code Review Best Practices

Andrejs Doronins describes code review as an amplifier because it amplifies both the good and the bad aspects of your application during the software development process. A **pull request (PR)** is a request to merge your code into another branch, which can have both benefits and drawbacks, depending on the situation.

This video will help you establish a process for doing code reviews while providing tips to review effectively. It also discusses the benefits of small pull requests and the downsides of big pull requests and describes the particularities of pull request feedback review comments and how to handle challenging code review situations.

Watch Code Review: Best Practices (01:22:00) by Andrejs Doronins to learn more about the importance of code reviewing.

Secure Software Testing for CSSLP

As you learn more about developing a test strategy, consider how this helps in tracking the test case and test scenarios in a test plan. The information in this video will discuss nonfunctional tests that cover the following:

- **Infrastructure**
- **Operating environments**
- **Performance**
- **Reliability**
- **Scalability**

The most important aspect of testing is test data, but it can be a challenge to ensure that the test data is complete, thorough, and accurate. The test data also must be stored securely and in a way that will maintain its integrity.

Watch the sections "Course Overview" (00:01:27), "Developing and Acquiring Test Data" (00:31:55), and "Executing the Test Plan" (00:23:01) from *Secure Software Testing for CSSLP* to learn more about developing test data and executing a plan.

Writing Custom Scripts

OWASP Zed Attack Proxy is one of the most commonly used open-source security tools. Therefore, the ability to script and program using this tool will elevate any software security developer. As you watch this video, you will review topics such as setting up the environment for OWASP ZAP using the Firefox browser with add-ons and the need for scripting to be able to perform customized testing as per users' requirements. Demos will be provided on how to check for request-response tampering, scripting authentication scenarios, generating custom payloads, and regressing security vulnerabilities.

Watch Writing Custom Scripts for OWASP Zed Attack Proxy (02:55:00) to explore more about writing custom scripts.

SonarQube is an open-source platform that can perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities in over 25 programming languages. As you watch this video, you will learn more about how SonarQube helps the DevOps team generate a high-quality base of code and how the components interact to perform static code analysis. You will explore the steps in installing and configuring SonarQube with a sample project running the analysis and reviewing the results. At the end of the video, resources will be provided for finding information related to SonarQube plug-ins and documentation.

Watch Application Analysis with SonarQube (00:34:00) for more information.

References

Gerlach, S. (13 minutes). *Modern dynamic application security testing* [Video]. PluralSight. https://app.pluralsight.com/library/courses/devseccon24-modern-dynamic-application-security-testing/table-of-contents

Doronis, A. (2021). *Code review: Best practices* [Video]. PluralSight. https://app.pluralsight.com/library/courses/code-review-best-practices/table-of-contents

Gunasekaran, M. (2019). *Writing custom scripts for OWASP Zed Attack Proxy* [Video]. PluralSight. https://app.pluralsight.com/library/courses/writing-custom-scripts-owasp-zed-attack-proxy/table-of-contents

Henry, K. (2020, Nov 19). *Secure software testing for CSSLP®* [Video]. PluralSight. https://app.pluralsight.com/course-player?clipId=eca4bc1d-0432-4b71-8ee6-dc871c5d0ad3

Ransome, J., Misra, A. and Merkow, M. (2023). *Practical core software security: A reference framework*. CRC Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=3298314&site=eds-live&scope=site&authtype=sso&custid=ns017578&ebv=EB&ppid=pp_1

Smight, G. (2022, Sept 26). *Application analysis with SonarQube* [Video]. PluralSight. https://app.pluralsight.com/library/courses/sonarqube-application-analysis/table-of-contents

Take a moment to think about what you learned in this lesson.

- During the A4 phase, any policy that exists outside the domain of the SDL policy is reviewed.
- Quality assurance testing occurs throughout the entire SDLC process.
- The three specific test type categories are benchmarks, scheduled tests, and exploratory tests.
- Code review finds and fixes a large number of security issues before the code is tested or shipped.
- The four basic techniques for code review are automated scanning, manual penetration testing, static analysis, and manual code review.
- AppSec describes finding, fixing, and preventing vulnerabilities at the application level.
- AppSec is difficult to scale for large organizations.
- The goal of code review is to catch bugs early on to decrease the cost of fixing them.
- Proxy scripts are effectively used to communicate a web security bug or web security control.
- Active and passive scanner scripts identify common vulnerabilities that are specific to your application.
- SonarQube gives developers the ability to continuously inspect the quality of code they produce.

Key Terms

- **abstract syntax tree (AST)**: the basis for software metrics and issues to be generated at a later stage
- **active scanner**: modifies the hypertext transfer protocol secure (HTTPS) inputs and analyzes the response to identify vulnerabilities
- **AppSec**: the process of finding, fixing, and preventing security vulnerabilities at the application level
- **benchmarks**: tests used to compare estimates to actual results
- **code review**: a process done to identify security vulnerabilities during software development

- **control flow analysis**: the mechanism used to step through logical conditions in the code
- **data flow analysis**: the mechanism used to trace data from the points of input to the points of output
- **Design and Development (A4) phase**: the fourth phase of the security development life cycle, in which you will build onto the proper process of security testing and continue to analyze necessities at the security level
- **documentation**: details and guides that are necessary to support the ongoing use of the software
- **dynamic analysis**: application security testing to identify vulnerabilities within a product application
- **exploratory tests**: done by the development tester to continually assess the quality of his or her work
- **Open Source Security Testing Methodology Manual**: a manual that provides templates and standards used when developing a test strategy
- **OWASP Zed Attack Proxy**: an open-source security tool used widely by software security developers
- **passive scanner**: silently analyzes all the hypertext transfer protocol (HTTP) requests and responses passing through the web application security tool
- **pull request**: a request to merge your code into another branch
- **scheduled tests**: mandatory requirements testing to validate the security of the software and associated system(s)
- **SonarQube**: open-source platform for static code analysis that can detect bugs, code smells, vulnerabilities, and hotspots in over 25 programming languages.
- **spider**: identifies inputs and supplies those to the scanning components of the security tool
- **static analysis**: analysis of computer software that is performed without actually executing programs
- **Zed Attack Proxy (ZAP)**: free, open-source penetration-testing tool

🔐 D487 - Software Testing & Deployment (Lessons 8–10)

Phase Covered: Ship (A5) + Post-Release Support
Competency: Ensuring product readiness & maintaining security post-release

---

🧪 Lesson 8: Determining Product Readiness

Goal: Ensure all **final security tasks** are completed before shipping.

Task	Description
🛡️ Final Security Review	Comprehensive check for policy compliance and threat closure
🛡️ Vulnerability Scan	Run tools to find known weaknesses (e.g., outdated libs, misconfigs)
🛡️ Penetration Testing	Simulate attacks to test product resilience
🛡️ Privacy Review	Ensure data collection, storage, transfer complies with **privacy laws**
🛡️ Open-Source Licensing Review	Check for OSS license conflicts (GPL, MIT, etc.) before release

✅ **Output:**
✔️ Product marked as **ready to ship**
✔️ Final signoff by **Security & Product Teams**

🛠️ Lesson 9: Post-Release Security Issues

Goal: Handle vulnerabilities and incidents **after launch**

Activity	Purpose
🛡️ Incident Response Planning	Predefine what to do when a breach or bug is reported

Activity	Purpose
Vulnerability Disclosure Policy	Tells customers & researchers how to report security bugs
Patch Management Process	Ensures fast deployment of security fixes to users
Customer Communication	Must be **clear and prompt** if vulnerabilities affect data or functionality

Real-world Tip:
Your team must act fast and responsibly — this is where your **reputation is tested.**

Lesson 10: Adapting to Security Frameworks and Models

Goal: Make sure your process aligns with industry-recognized models.

Framework/Model	Why It Matters
NIST Cybersecurity Framework	Defines secure practices for identify, protect, detect, respond, recover
OWASP Top 10	Must be checked pre- and post-release; addresses **most common vulnerabilities**
Microsoft SDL	WGU course SDL model — aligns with Ship/Post-Release phases
ISO/IEC 27001	Used for compliance/security maturity in larger orgs

---

🐘 Final Exam Hot List

✅ Know these for sure:

- 🐘 Final Security Review components (scan, pentest, OSS license check, privacy)
- 🐘 Post-release vulnerability handling process
- 🐘 Incident response basics
- 🐘 Security frameworks: NIST, OWASP Top 10
- 🐘 Ship = last check before release
- 🐘 Post-Release = response, patch, communicate

🧠 Quick Visual: Ship vs Post-Release

SDL Phase	Key Activities
🐘 **Ship (A5)**	Final reviews, scans, testing, signoff
🐘 **Post-Release**	Incident response, patching, customer communication, vulnerability management

🛠️ D487 Section 4 – Build Employability Skills & Competence

🚀 Why This Matters (Career-wise)

Cybersecurity threats are evolving. Employers **need developers who can build AND secure software**.

🐘 **You will be tested** on real-world readiness:
Can you build, test, and release secure software?

🔑 Skills Covered in This Section

🧠 Skill	🐘 On Final?	💡 What It Means
Develop standards/methods for secure development	🐘 Yes	Create checklists or practices to ensure code is safe, reliable, and release-ready
Evaluate mitigation/remediation based on risk	🐘 Yes	Choose the right fix for a vulnerability depending on how bad the risk is
Determine incident response steps and root cause	🐘 Yes	Know what to do when a security issue occurs and why it happened
Apply software roles within an agile development environment	🐘 Yes	Understand how security fits into sprints, standups, and agile team roles

---

✅ What You Need to Know for the Final Assessment

Prep Question	🐘 Must Know?	✔️ Quick Explanation
Can I ensure that my software or application is ready to be released?	🐘 Yes	Know the **Ship Phase** tasks (Final security review, OSS review, etc.)
Can I identify the best techniques to identify vulnerabilities of my software or application?	🐘 Yes	Master **static analysis, dynamic analysis, fuzzing, manual review**
Can I complete thorough vulnerability scanning to ensure that my software or application is secure?	🐘 Yes	Use automated tools like SonarQube, OWASP ZAP, Nessus, etc.
Can I use the security development framework to ensure that my software is secure?	🐘 Yes	Understand the SDL phases and how security is embedded throughout

---

🧪 Assessment Tip Breakdown

Topic	What You Should Know
🦣 SDL Model	Each phase (especially Ship + Post-Release), roles, artifacts, goals
🦣 Agile + Security	How dev roles like Scrum Master or Product Owner tie into SDL
🦣 Threat Modeling	STRIDE model basics; how to identify threats at design time
🦣 Testing Techniques	Pros/cons of Static vs. Dynamic vs. Fuzz vs. Manual reviews
🦣 Vulnerability Management	How to rank, triage, and respond to vulns based on risk level
🦣 Incident Response	Steps to take post-breach: detection → mitigation → disclosure → patch

---

🧠 **Test Readiness Self-Check (must answer YES to all)**

- ☐ I know how to prepare software for secure release.
- ☐ I can evaluate and select the right vulnerability scanning techniques.
- ☐ I understand mitigation vs remediation.
- ☐ I can map SDL and Agile frameworks together.
- ☐ I can confidently explain what happens **after release** (post-release support).

🦣 What Happens in the Ship (A5) Phase?

Activity	Purpose
🦣 Final Policy Compliance Analysis	Confirms the software meets **internal policies, industry standards, and laws**
🦣 Final Security Review	Ensure that **all security risks are addressed** before release
🦣 Final Privacy Review	Check for **PII compliance**, data handling, and privacy laws (e.g., HIPAA, GDPR)
🦣 Final Security Testing Plan	Plans out tests like **pen tests, fuzzing**, and scan verification before go-live
🦣 Security Communication Plan	Define how support and customer service are informed of **security concerns**
🦣 Release Approval Decision	Based on all findings above — if all are clean, the product is **approved for shipping**

🧠 Learning Objective 1: Evaluate Criteria for Product Release

To approve a release, all of the following must be **successfully completed**:

- ✔ Final security & privacy review
- ✔ Policy compliance pass
- ✔ No critical unpatched vulnerabilities
- ✔ Risk level is acceptable (documented & mitigated)
- ✔ Customer support team is trained on known security issues

🦣 This will be on the final exam — know what counts as "release ready."

🧠 Learning Objective 2: Identify a Strategy or Technique to Remediate Vulnerabilities

🦣 Remediation Technique	Example
Input validation	Filter/sanitize all user inputs (e.g., form fields)

🐘 Remediation Technique	Example
Patch vulnerable libraries	Update open-source dependencies with security patches
Replace insecure authentication	Use MFA or OAuth2 instead of simple login forms
Apply least privilege	Restrict access to sensitive data and code modules
Implement encryption at rest/in-transit	Protect data in storage and during communication

🐘 Final test will include questions where you must **choose the correct mitigation strategy** for a given scenario.

✅ **Summary for Test Prep**

Topic	🐘 On Test	💡 Know This
Final SDL Ship Tasks	🐘 Yes	Security review, privacy check, policy analysis, testing plan
Release Approval Criteria	🐘 Yes	All issues mitigated or accepted, support team informed
Remediation vs. Mitigation Techniques	🐘 Yes	Be able to match fix strategy to vulnerability scenario
Communication Plan Importance	Maybe	How support teams get info post-launch

Ship (A5)

The **Ship (A5) phase** of the security development life cycle takes place in the last phase of the software development life cycle when the organization is preparing to release the product. Security teams perform policy compliance analysis, vulnerability scans, penetration testing, open-source licensing reviews, a final security review, and a final privacy review to determine whether the product is ready to release to customers.

Read chapter 7 "Ship (A5): SDL Activities and Best Practices" from *Practical Core Software Security: A Reference Framework* for more information on this phase of the SDLC.

Conducting Network Vulnerability Analysis

Conducting network vulnerability analysis is the process of discovering, identifying, and classifying vulnerabilities and how much of a threat they are to customers. This is crucial to the SDLC, as it impacts the SDL immensely. As network vulnerability analysis is conducted, developers need to evaluate threats and begin to incorporate solutions into their applications. This portion of the lesson covers preparing an analysis toolkit, getting into the hacking mindset, harvesting data using internet resources, and executing vulnerability scans.

Watch Conducting Network Vulnerability Analysis (02:28:00) to learn more.

Assessing the Impact of Web Application and Vulnerabilities

Web applications can be very susceptible to attack due to their availability and the way they communicate. This lab demonstrates how to assess site and database functionality and how to test vulnerabilities, concentrating on SQL injection and authentication.

Complete Assessing the Impact of Web Application and Vulnerabilities from Infosec Learning.

Analyzing Output from Web Application

SQL injection and authentication are two of the most prevalent security threats to modern web applications, but they are not the only ones. This lab demonstrates how to analyze output from web applications. It covers configuration of an intercept proxy, inspecting header and session data, testing command injection, submitting fuzzy input, and running a vulnerability scanner.

Complete Analyzing Output from Web Application Assessment Tools from Infosec Learning.

References

Analyzing output from web application assessment tools. (2023). Infosec Learning, Inc. https://lab.infoseclearning.com/course/PUVFALSIHK/lab/PVYKKYMZIC?check_logged_in=1

Assessing the impact of web application vulnerabilities. (2023) Infosec Learning, Inc. https://lab.infoseclearning.com/course/PUVFALSIHK/lab/ODGYGUDDQR?check_logged_in=1

Cardwell, K. (2017). *Conducting network vulnerability analysis* [Video]. PluralSight. https://app.pluralsight.com/library/courses/network-vulnerability-analysis-conducting/table-of-contents

Ransome, J., Misra, A., and Merkow, M. (2023). *Practical core software security: A reference framework*. CRC Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=3298314&site=eds-live&scope=site&authtype=sso&custid=ns017578&ebv=EB&ppid=pp_1

LESSON 8

Take a moment to think about what you learned in this lesson.

- The Ship (A5) phase of the SDLC occurs when the security team performs its final analysis and security review on the applications or software.
- Policy compliance analysis verifies the product meets quality standards before the release of an application or software.
- Vulnerability scanning tools attempt to identify weakness in the applications.
- Penetration testing simulates the actions of a hacker to attempt to identify vulnerabilities within the software.
- The four phases of penetration testing are: assess, identify, evaluate and plan, and deploy.
- Active and passive analysis techniques are both useful during vulnerability testing.
- Creating a networking laboratory allows you to test within a controlled environment without written authorization and permissions.
- NMap is a popular network scanning tool.
- There are many techniques that can be used to discover vulnerabilities; it is important to consider which technique will be best for your software or application.

Key Terms

- **authenticated scans:** scans that require software to log onto a system to scan it
- **external scans:** scans that target security issues that are found outside the firewall
- **internal scans:** scans to identify security issues that a malicious attacker could exploit from inside the network
- **intrusive target search:** scans to exploit a vulnerability when it is identified
- **Nmap:** a tool used for network scanning and security auditing
- **open-source software license compliance:** regulations regarding the software licensing of in-house products
- **open-source software security:** identifying software security within in-house developed software
- **penetration testing:** an authorized attack of an application to determine its weaknesses
- **range:** a networking laboratory created to conduct vulnerability analysis testing
- **Ship (A5) phase:** the fifth phase of the security development lifecycle that verifies that the product complies with security policies
- **SQL injection:** a code injection that might destroy your software
- **target machine:** a virtual space to practice identifying attack surfaces of the machine
- **virtualization:** technology used to create software services
- **vulnerability scan:** explore application and databases to attempt to identify weaknesses
- **vulnerability sites:** websites with information on the latest known vulnerabilities

📦 What Are PRSAs?

Post-Release Support Activities (PRSA) are **security tasks performed after the product has been released** to customers. These activities **maintain and monitor security** as users interact with the application in the real world.

📜 Key PRSA Deliverables

Deliverable	Purpose
🦣 **External Vulnerability Disclosure Report**	Public-facing process to disclose security issues found post-release.
🦣 **Third-Party Security Reviews**	External auditors verify that the application meets **compliance/security standards**.
🦣 **Post-Release Certifications**	Formal certifications (e.g., ISO/IEC 27001) confirming security posture.
🦣 **Internal Reviews (New Combos/Cloud)**	Evaluate **new deployments** like cloud hosting or product integrations.
🦣 **Security Architecture Reviews**	Reassess system architecture after release or platform changes.
🦣 **Tool-Based Assessments**	Continuous scanning/monitoring tools to detect vulnerabilities in production.

🎯 **Learning Objective: Determine the Cause and Response to a Post-Release Security Incident**

🐘 **Steps in Post-Release Incident Response**

1. **Detection**
    - Use **monitoring tools**, logs, or customer reports to identify anomalies or breaches.
2. **Analysis**
    - Determine the **root cause**: Was it a coding flaw? A configuration issue? A credential leak?
3. **Containment**
    - Isolate the affected system or user accounts to stop the damage.
4. **Mitigation / Remediation**
    - Apply hotfixes, patches, or configuration changes.
    - Roll out **updated versions** or disable vulnerable features temporarily.
5. **Notification**
    - 🐘 Inform customers, partners, and compliance bodies **as required by law or policy** (e.g., GDPR, HIPAA).
6. **Postmortem / Documentation**
    - Record lessons learned, update policies, and strengthen tools or team response.

🦣 Final Exam Focus

You **will be tested** on:

- 🦣 What PRSA tasks happen after a product is released
- 🦣 What documents/reviews/certifications are involved
- 🦣 How to identify and respond to a **post-release vulnerability or security incident**
- 🦣 The order of response: Detect → Analyze → Contain → Remediate → Notify → Document

---

✅ Summary Table: PRSA Breakdown

Category	Key Task
🦣 Disclosure	External vulnerability report
🦣 Review & Audit	3rd party review, internal review, architecture analysis
🦣 Continuous Monitoring	Tool-based assessments post-release
🦣 Incident Response	Determine cause, patch, notify, and document the event

Post-Release Support

The **Post-Release Support phase** of the security development life cycle is where organizations prepare for vulnerabilities that are identified after the product has been released to customers, including vulnerabilities found in both managed and unmanaged codebases.
Key stakeholders are identified, customer communication plans are established, and a process for handling newly discovered vulnerabilities is defined.

Read chapter 8 "Post-Release Support (PRSA1-5)" from *Practical Core Software Security: A Reference Framework* to learn more.

Reference

Ransome, J., Misra, A., and Merkow, M. (2023). *Practical core software security: A reference framework*. CRC Press.
https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=3298314&site=eds-live&scope=site&authtype=sso&custid=ns017578&ebv=EB&ppid=pp_1

LESSON 9

Take a moment to think about what you learned in this lesson.

- Having software security experts report to the engineering organization allows for a stronger relationship during the software security development process.
- Quality security is built throughout the entire engineering process, not in just one phase of the development life cycle.
- Not every company will be able to include all PRSAs, so consider which has the highest value to your organization and how to optimize the tools you do have access to.
- CVSS is a model that is used to assess the severity of a vulnerability.
- Post-release privacy issues may need additional development, quality assurance, and/or security resources.
- Third-party reviews may be necessary when completing a post-release review.
- During a company's merger or acquisition, software security may go under architectural review to identify any changes that will need to take place once the merger or acquisition is complete.

- Requirements for post-release certifications should be included in security and privacy requirements before deployment.

Key Terms

- **Common Vulnerability Scoring System (CVSS):** a model used to assess the severity of a vulnerability
- **legacy code:** old code that is no longer supported
- **merger and acquisition (M&A):** when companies consolidate
- **Product Security Incident Response Team (PSIRT):** the team that receives, investigates, and reports security vulnerabilities
- **Post-Release Support phase:** the phase of the SDLC in which organizations prepare for vulnerabilities after the product has been released
- **Post-Release PSIRT Response:** responds to software product security incidents that involve the external discovery of post-release software vulnerabilities
- **Software Security Champion (SSC):** an expert on promoting security awareness, best practices, and simplifying software security
- **Software Security Evangelist (SSE):** an expert to promote awareness of products to the wider software community

🏗️ D487 – Adapting SDL to Organizational Environments

🎯 Lesson Goal:

Learn how to **integrate the SDL** into modern environments like **Agile** and **DevSecOps**, and recognize how deployment models and security maturity affect SDL execution.

🦣 Top 4 Environments for Deployment

Environment	What to Know
🦣 **On-premises**	Internal control, but requires full SDL responsibility
🦣 **Cloud**	Focus on shared responsibility model (provider handles infra, you secure code)
🦣 **Hybrid**	Combines on-prem and cloud — adds complexity to threat modeling & policy
🦣 **Mobile/IoT**	High risk: small devices, weak controls, frequent updates needed

🧠 *You must know which environment affects which security choices — especially in threat modeling, testing, and deployment.*

🦣 Key Success Factors by SDL Phase

Phase	Key Deliverables / Success Factors
🦣 **Requirements**	Risk rating, compliance checklists, security goals
🦣 **Design**	Threat models, data flow diagrams, STRIDE mapping
🦣 **Implementation**	Secure coding standards, peer review, static code analysis
🦣 **Verification**	Security testing (DAST, fuzzing), tool results reviewed
🦣 **Release (Ship)**	Final review, open-source license check, policy compliance

Phase	Key Deliverables / Success Factors
Post-Release	Incident response plan, vuln management, tool-based monitoring

---

Adapting SDL to Agile

SDL Element	How it fits in Agile
Security stories	Add to backlog alongside user stories
Security tasks per sprint	Break SDL into sprint-aligned tasks (e.g., threat modeling per feature)
Security Champion Role	Internal security advocate in each scrum team
Secure CI/CD Pipeline	Automated scans at each integration point

*Agile ≠ No security. It just means **shorter, iterative SDL cycles.***

Adapting SDL to DevSecOps

DevSecOps Practice	SDL Integration
Automated testing	Integrate SAST/DAST in CI/CD
Continuous compliance	Enforce policy gates at build & deploy
Shift-left security	Apply security as early in development as possible
Security as code	Define policies, scans, and rules in version-controlled configurations

---

📦 Software Security Maturity Models

Model	Purpose
🐘 **BSIMM**	Benchmarks orgs on 12 security practices (industry standard)
🐘 **OpenSAMM**	Open-source model to assess & improve software security
Microsoft SDL Maturity	Measures how well SDL practices are integrated org-wide

*Know that maturity models help teams assess **how far along** they are in securing software development*.

🧠 Final Exam Must-Know Summary

🐘 **Concept**	**Why it Matters**
SDL adapts to **Agile** via backlog, sprints, champions	You'll see this exact adaptation on the test
SDL adapts to **DevSecOps** via automation & CI/CD	Must know how testing tools and policies integrate into build pipelines
Know the **4 deployment environments** and their risks	Will be asked to choose how environment affects SDL or threat modeling
Understand **software maturity models**	May be asked to compare BSIMM vs OpenSAMM or how they measure org progress

Post-Release Support

Implementing organizational security policies does not happen overnight and may require negotiation with key stakeholders and organizational leadership. Adapting a reference framework can help navigate those challenges by relying on proven, tested SDL activities, deliverables, and methodologies. There is no reason to reinvent the wheel when there are industry-wide accepted best practices available.

Read chapter 9 "Adapting Our Reference Framework to Your Environment" from *Practical Core Software Security: A Reference Framework* to gain a deeper understanding on reference frameworks.

Reference

Ransome, J., Misra, A. and Merkow, M. (2023). *Practical core software security: A reference framework*. CRC Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=3298314&site=eds-live&scope=site&authtype=sso&custid=ns017578&ebv=EB&ppid=pp_1

LESSON 10

Take a moment to think about what you learned in this lesson.

- Your software is most likely to be deployed in Agile, DevOps, Digital Enterprise, or a combination of those environments.
- Agile development is designed to deliver value faster.
- DevOps teams work together for ongoing operations, enhancements, defect removal, and optimization of resources.
- Cloud technology has created a rethinking on how applications are built, deployed, and used.
- Digital enterprises use technology to assist in enabling and improving business activities.
- Moving to public cloud services has increased security challenges.
- OpenSAMM business functions include governance, construction, verification, and deployment.
- Building Security in Maturity Model (BSIMM) is a study of existing software security initiatives used to gather data from larger software development.

- The four types of BSIMM categories are governance, intelligence, software security development life cycle touchpoints, and deployment.
- Companies can use BSIMM documents to conduct their own assessments.
- Threats can be classified using the STRIDE acronym: spoofing, tampering, repudiation, information disclosed, denial of service, and elevation of privilege.

Key Terms

- **code review (CR):** a practice of verification involving review of an organization's secure code to identify vulnerabilities
- **construction:** a function of OpenSAMM centered around how organizations define goals and create software within development projects
- **deployment:** a function of OpenSAMM centered around how an organization releases software
- **design review (DR):** a practice of verification involving inspecting artifacts that were created from the design process
- **digital enterprise:** technology used to enable and improve business activities
- **education and guidance (EG):** a practice of governance involving increasing security knowledge among software developers
- **environment hardening (EH):** a practice of deployment involving implementing controls for the operating environment of an organization's software
- **governance:** a function of OpenSAMM centered on how organizations manage overall software development activities
- **Open Software Assurance Maturity Model (OpenSAMM):** an open framework to help organizations implement software security tailored to the organization's specific risks
- **Operational Enablement (OE):** a practice of deployment involving identifying and capturing security-relevant information
- **policy and compliance (PC):** a practice of governance involving setting up security and compliance control

- **secure architecture (SA):** a practice of construction involving activities to prompt secure-by-default designs during the design process
- **security requirements (SR):** a practice of construction involving the promoting of inclusion security requirements during software development
- **security testing (ST):** a practice of verification involving testing the organization's software in its environment
- **strategy and metrics (SM):** a practice of governance involving the strategies regarding software assurance and processes to collect metrics on an organization's security
- **threat assessment (TA):** a practice of construction involving identifying and characterizing potential threats
- **verification:** a function of OpenSAMM centered around how an organization checks and tests artifacts produced through software development
- **vulnerability management (VM):** a practice of deployment involving establishing processes for managing internal and external vulnerability reports

🩹 **D487 – Chapter 10: *Adapting to Security Frameworks and Models***

🐘 **Chapter Focus:**

How to **align the Security Development Lifecycle (SDL)** with:

- Organizational practices (Agile, DevSecOps)
- Industry-recognized **security frameworks**
- Security **maturity models**

✅ **Key Takeaways (What You Must Know)**

Concept	Summary
🐘 **SDL Adaptability**	SDL can and must be adapted to **Agile**, **DevOps**, **Cloud**, **Mobile**, etc.

Concept	Summary
🦣 **Security Frameworks**	Use frameworks (like **NIST**, **OWASP**, **Microsoft SDL**) to guide practices
🦣 **Security Maturity Models**	Tools like **BSIMM** and **OpenSAMM** measure an org's SDL maturity
🦣 **Threat Modeling Improvements**	Enhancing threat models improves risk discovery & solution planning

---

🔁 SDL in Real-World Development Models

🦣 Agile + SDL Integration

- Security is broken down into **backlog tasks**
- Add **threat modeling and code reviews** into each sprint
- Assign a **security champion** per Agile team

🦣 DevSecOps + SDL Integration

- Automate scans (SAST, DAST) in the **CI/CD pipeline**
- Include **security policy gates** before deployment
- Use "**security as code**" for consistency and traceability

---

🏗️ 4 Key Deployment Environments

Environment	Security Focus
🦣 On-premises	Full control but **full SDL responsibility**
🦣 Cloud	**Shared responsibility model** with provider
🦣 Hybrid	Combines on-prem + cloud; higher complexity & **threat modeling needed**

Environment	Security Focus
Mobile/IoT	Frequent updates, **limited controls**, and high exposure risk

---

Maturity Models to Know

Model	Purpose
BSIMM	Benchmarks your org against **12 practices** of mature secure dev
OpenSAMM	Open-source tool to evaluate and improve software security posture
Microsoft SDL Maturity Model	Internal Microsoft standard, focuses on process integration

---

Security Frameworks to Align With

Framework	Why It's Important
NIST Cybersecurity Framework (CSF)	Covers Identify → Protect → Detect → Respond → Recover
OWASP Top 10	List of most common web app vulnerabilities (e.g., XSS, SQLi, A01–A10)
Microsoft SDL	The model WGU teaches—important to compare with others

---

Final Exam Tips for Chapter 10

✅ Know how to **integrate SDL into Agile and DevSecOps**
✅ Be able to match **maturity models** (BSIMM, OpenSAMM) to their purpose
✅ Be able to identify how **deployment environments** affect SDL
✅ Expect a question on the **shared responsibility model** (especially in cloud)
✅ Study the **phases of NIST CSF** and **OWASP Top 10 relevance to testing phase**

🔐 D487 – Software Security After Release (Final Phase of SDL)

🦣 Security Work Doesn't Stop at Release

Just because the code is shipped doesn't mean your security job is over.
Both **pre-release** and **post-release** phases are part of a secure software lifecycle.

✅ Pre-Release Must-Haves (Before Shipping Product)

Task	Purpose
🦣 Final SDL Checkpoint	Ensure ALL SDL steps were completed before go-live
🦣 Framework & Regulatory Compliance	Validate compliance with standards (e.g., **HIPAA**, **GDPR**, **NIST**)
🦣 Final Vulnerability Scan	Use automated tools to catch known flaws
🦣 Penetration Testing	Simulate attacks to test for **real-world exploitation**
🦣 Final Security & Privacy Review	Confirm secure handling of PII, consent, and legal data storage practices

---

🛠️ Post-Release Responsibilities (Support Phase)

Once the product is released, the organization **must maintain a repeatable and responsive process** to continue security protections.

Post-Release Activity	Why It Matters
🦣 Incident Response Plan	Required to **quickly detect and mitigate threats** that surface post-release
🦣 Customer Communication Process	Must notify users when their data or security might be impacted
🦣 Ongoing Vulnerability Monitoring	Continuously **scan and monitor** for newly discovered flaws or exploits

Post-Release Activity	Why It Matters
🦣 Patch & Remediation Pipeline	Rapid response to fix known bugs or vulnerabilities
🦣 Disclosure Policy	Clear external policy for researchers to report vulnerabilities

---

🦣 Final Exam Hot Points

You WILL be tested on:

- 🦣 SDL activities that must be completed **before release**
- 🦣 What needs to happen **after release** (monitoring, response, patching)
- 🦣 The difference between **vulnerability scans** vs. **pen testing**
- 🦣 The need for **regulatory compliance** and **privacy checks**
- 🦣 Having a repeatable, documented **post-release process**

🧠 Quick Tip:

Use this mental checklist when reviewing **release readiness**:

✅ Compliance = Met
✅ Security = Reviewed
✅ Privacy = Verified
✅ Pen Test = Completed
✅ Vulnerabilities = Scanned & resolved
✅ Incident Plan = Documented
✅ Communication Channels = Ready

COMPLETE CHAPTER REVIEW OF ALL CHAPTERS FROM THE BOOK

🐘 D487 Secure Software Design – Final Exam Chapter Overview

📘 Chapter 1: Introduction to Software Security

- 🐘 Importance of building security in from the start
- 🐘 Difference between quality vs. security in code
- 🐘 Overview of the Security Development Lifecycle (SDL)
- 🐘 CIA Triad: Confidentiality, Integrity, Availability
- 🐘 When to apply threat modeling and risk analysis
- 🐘 SDL vs. SDLC comparison

📘 Chapter 2: Core Concepts of Secure Software Design

- 🐘 STRIDE threat modeling categories
- 🐘 Data flow diagram (DFD) components:
  - External elements = rectangles
  - Data store = parallel lines
  - Data flow = solid line + arrow
  - Trust boundary = dashed line
- 🐘 Risk rating methodologies (DREAD, CVSS)
- 🐘 Security objectives and business goals alignment

📘 Chapter 3: The Requirements Phase

- 🐘 Identify security objectives early
- 🐘 Include regulatory requirements (e.g., HIPAA, GDPR)
- 🐘 Privacy Impact Rating scale (P0–P3) — P1 = High Risk if handling PII

- 🦣 **Use security checklists, legal policies, and business risk discussions**

📘 **Chapter 4: The Design Phase**

- 🦣 **Threat modeling as a mandatory SDL activity**
- 🦣 **Use STRIDE to model potential threats**
- 🦣 **Define trust boundaries and sensitive data paths**
- 🦣 **Document threat scenarios and rank risk levels**
- 🦣 **Assign mitigation strategies based on risk rating**

📘 **Chapter 5: The Implementation Phase**

- 🦣 **Apply secure coding standards (e.g., input validation, error handling)**
- 🦣 **Code reviews – must be done same day as development**
- 🦣 **Use static analysis tools (SAST) to find flaws early**
- 🦣 **Use secure libraries and frameworks**

📘 **Chapter 6: The Verification Phase**

- 🦣 **Dynamic Analysis (DAST) – run against the live app**
- 🦣 **Fuzz Testing – send random/malformed input to break stuff**
- 🦣 **Manual source code review – catches logic/design errors**
- 🦣 **Know the difference between static, dynamic, fuzz, and manual testing**
- 🦣 **Security Test Plan – must align with threats and risks**

📘 **Chapter 7: The Release (Ship) Phase**

- 🦣 **Final Security Review – last chance to catch flaws**

- 🦣 **Final Privacy Review – ensure compliance with data laws**
- 🦣 **Policy compliance analysis – sets release gates**
- 🦣 **Open-source licensing review**
- 🦣 **Pen testing and vulnerability scans must be completed**
- 🦣 **Communicate security issues to support teams**

📘 **Chapter 8: Post-Release Support Activities**

- 🦣 **External vulnerability disclosure process**
- 🦣 **Third-party security reviews**
- 🦣 **Post-release certifications (e.g., ISO/IEC 27001)**
- 🦣 **Internal reviews for new cloud deployments or integrations**
- 🦣 **Security incident response plan**
- 🦣 **Any architecture or code change must re-trigger SDL tasks**

📘 **Chapter 9: Assessing Security Readiness**

- 🦣 **Criteria for approving a secure release**
- 🦣 **How to prioritize and remediate vulnerabilities**
- 🦣 **Determine root cause of incidents**
- 🦣 **Risk-based mitigation approach**
- 🦣 **Incident response: Detect → Contain → Notify → Document**

📘 **Chapter 10: Adapting SDL to Frameworks**

- 🦣 **Integrating SDL with Agile: security stories, champions, sprint reviews**
- 🦣 **Integrating SDL with DevSecOps: automated scans, shift-left, CI/CD gates**

- 🐘 **Security maturity models:**
  - **BSIMM (industry comparison)**
  - **OpenSAMM (open-source improvement model)**
- 🐘 **Security frameworks:**
  - **NIST CSF (Identify → Protect → Detect → Respond → Recover)**
  - **OWASP Top 10**
  - **Microsoft SDL**

✅ **Quick Test Strategy:**

You Should Be Able To…

🐘 **Match SDL phases to activities and artifacts**

🐘 **Recognize security roles in Agile/DevOps**

🐘 **Distinguish between static, dynamic, and fuzz testing**

🐘 **Identify security responses and post-release support tasks**

🐘 **Analyze a threat scenario and recommend the right mitigation**

🐘 **Recall privacy risk levels and regulatory concerns**

🐘 **Pick the correct tool or framework for a given security need**

🧠 **D487 Final Exam – One Page Chapter Overview (Test-Only Focus)**

🦣 Chapter 1: Introduction

- 🦣 **CIA Triad: Confidentiality, Integrity, Availability**
- 🦣 **SDL vs SDLC differences**
- 🦣 **Security must be built-in from the beginning**
- 🦣 **Threat modeling and risk assessment start early**

🦣 Chapter 2: Core Design Concepts

- 🦣 **STRIDE threat model categories**
- 🦣 **Data Flow Diagram (DFD) elements:**
  - **Trust boundary = dashed line**
  - **External element = rectangle**
  - **Data flow = solid arrow**
  - **Data store = two parallel lines**
- 🦣 **Risk assessment = DREAD or CVSS**

🦣 Chapter 3: Requirements Phase

- 🦣 **Identify security and privacy requirements**
- 🦣 **Privacy Impact Ratings:**
  - **P1 = High risk (handles PII)**
- 🦣 **Include legal/regulatory compliance (e.g., HIPAA, GDPR)**

🦣 Chapter 4: Design Phase

- 🦣 **Perform threat modeling with STRIDE**
- 🦣 **Define trust boundaries, assets, and threats**

- 🦣 **Assign mitigations based on risk rating**

🦣 **Chapter 5: Implementation Phase**

- 🦣 **Follow secure coding standards**
- 🦣 **Peer code reviews done same day as code is written**
- 🦣 **Use static analysis tools (SAST)**

🦣 **Chapter 6: Verification Phase**

- 🦣 **Know tool types:**
  - **Static = code without running**
  - **Dynamic = code during execution**
  - **Fuzz = random data inputs**
  - **Manual review = logic & design flaw catching**
- 🦣 **Match tools to vulnerability types**

🦣 **Chapter 7: Ship Phase**

- 🦣 **Final policy compliance analysis = quality gates**
- 🦣 **Final security & privacy review**
- 🦣 **Vulnerability scanning + Penetration testing**
- 🦣 **OSS license review**
- 🦣 **Communicate risks to support teams**

🦣 **Chapter 8: Post-Release Support**

- 🦣 **External vulnerability disclosure process**
- 🦣 **Incident response: Detect → Contain → Patch → Notify**

- **SDL must restart for new deployments/code reuse**
- **Use tool-based assessments after release**
- **Third-party security reviews & certifications**

Chapter 9: Assessing Readiness

- **Evaluate if software is secure enough to ship**
- **Prioritize mitigation based on risk**
- **Patch vulnerabilities before they impact customers**
- **Root cause analysis of incidents**

Chapter 10: Frameworks & Models

- **SDL fits into Agile: use security backlog, security champions**
- **SDL fits into DevSecOps: CI/CD security, automation**
- **Maturity models:**
  - **BSIMM (benchmark practices)**
  - **OpenSAMM (improvement model)**
- **Know OWASP Top 10 + NIST CSF functions:**
  - **Identify, Protect, Detect, Respond, Recover**

✅ Quick SDL Phase Reference Cheat Sheet

SDL Phase	What Happens
🐘 Requirements	Define security & privacy goals, identify legal & compliance needs
🐘 Design	Threat modeling, STRIDE, DFDs, architecture risk analysis
🐘 Implementation	Secure coding, peer review (same day), static analysis (SAST)
🐘 Verification	DAST, fuzzing, manual reviews, full test plan execution
🐘 Release (Ship)	Final reviews, pen test, OSS review, compliance gates, support communication
🐘 Post-Release	Disclosure policy, incident response, 3rd party review, internal audits, maturity tracking

---

🔥 High-Yield Topics That Repeat on the Test

- 🐘 **Code Review Timing: Must be completed within hours or same day after code is written**
- 🐘 **Vulnerability in URL like ?category=2 OR 1=1: = SQL Injection**
- 🐘 **Final quality gates are defined in: A5 Policy Compliance Analysis**
- 🐘 **If anything changes post-release (cloud migration, reuse, API updates): You MUST restart SDL**

🔐 Security Tool Match-Up (Test Trap Section)

Tool Type	Purpose
🐘 Static (SAST)	Scans source code without execution (early phase, automation friendly)
🐘 Dynamic (DAST)	Scans running app, finds runtime bugs (auth issues, input flaws)
🐘 Fuzzing	Random/invalid data input to test crash/overflow/edge cases
🐘 Manual Review	Human eyes catch logic/design flaws tools can't

---

📦 Testing Types You Need to Know

Testing Type	When Used	Example
🐘 Alpha Testing	Internal testing before beta	Done by developers or QA
🐘 Beta Testing	By real users post-release	Customers test before wide rollout
🐘 Unit Testing	Code-level, function-by-function	Developer checks small code units
🐘 Integration Testing	Connecting multiple components	APIs, database calls, services

---

📚 Regulations & Frameworks to Memorize

Item	Why It Matters
🐘 HIPAA/GDPR	Must be addressed in privacy reviews
🐘 NIST CSF	5 functions: Identify, Protect, Detect, Respond, Recover
🐘 OWASP Top 10	Standard list of common vulnerabilities (XSS, Injection, Broken Access)
🐘 BSIMM/OpenSAMM	Maturity models that assess how "security mature" your SDL is

---

🧠 Tips to Avoid Mistakes on the Test

- ❌ Don't confuse *mitigation* (reduce severity) with *remediation* (fix it)
- ❌ Don't mix up *static* (pre-execution) and *dynamic* (runtime) testing
- ✅ Always choose the answer that includes legal/privacy/compliance AND security — not just one
- ✅ When in doubt, assume security must be continuous even after release
- 🔄 Don't Confuse These SDL Activities
-

📄 Policy & Compliance Must-Know

- 🐘 Policy Compliance Analysis (Ship/A5 Phase) = defines quality gates
- 🐘 If a release doesn't meet policy gate: product can't ship
- 🐘 Open-source license review = required in Ship phase
- 🐘 Privacy review = checks for PII handling and laws like HIPAA, GDPR

Easily Confused	Correct Mapping
Code Review vs Static Analysis	Code review = human inspection; Static = automated tool
Penetration Test vs DAST	Pen test = simulated attacker; DAST = scanner analyzing runtime behavior
Threat Modeling vs Risk Rating	Threat modeling = what could go wrong; Risk rating = how bad & how likely it is
Mitigation vs Remediation	Mitigation = reduce/lessen risk; Remediation = fix the problem

Attack Types & Examples That Show Up

Attack	How It Shows Up
SQL Injection	**URL ends in ?id=1 OR 1=1**
Cross-Site Scripting	**Form or comment box injecting JavaScript (e.g., <script>alert(1)</script>)**
Man-in-the-Middle	**Intercepting unsecured traffic between client and server**
Broken Access Control	**Users seeing/editing data they shouldn't have access to**

Vulnerability Reporting = Post-Release

- **Use a vulnerability disclosure policy**
- **Must include contact, response time, and fix timeline**
- **SDL must restart if post-release changes are made (e.g., new feature or cloud move)**

🚨 Final Exam Traps to Watch For

- ❌ If you see “done after release = done with SDL” → WRONG. SDL continues post-release
- ❌ “Static analysis is the same as manual review” → WRONG
- ✅ If an answer includes compliance, privacy, AND security → That’s your best choice
- ❌ If there’s no final pen test or scan before release → Reject release

🧠 Key Tools to Recognize

Tool	What It Does
SonarQube	Static code analysis (SAST)
Jenkins	CI/CD automation server
JIRA	Issue/bug tracking
Dynatrace	AI-based performance monitoring & security

✅ Final Sanity Checklist Before You Test:

- Know all SDL phases and what happens in each
- Can tell the difference between SAST, DAST, fuzzing, manual review
- Understand STRIDE & DFD diagram symbols
- Recognize real attack examples (XSS, SQLi)
- Know what happens post-release (support, incident response, disclosure)
- Understand how SDL works inside Agile & DevSecOps
- Familiar with frameworks: OWASP, NIST, HIPAA, GDPR
- Can match tools to tasks (e.g., SonarQube → SAST)

✅ Appendix A: Case Study for Chapters 3–8 Exercises

Context Summary:

- The case study is centered around a fictional company **Revvin' Engines**, an e-commerce platform that suffered a **major security breach**.
- Attackers exploited **SQL injection vulnerabilities**, gaining full access to **credit card databases**.
- Logs reviewed: web logs, server logs, application logs, and Windows Event logs.
- Critical vulnerabilities:
- Lack of secure coding practices
- Inadequate access controls
- Use of dynamically generated SQL queries
- The aftermath included:
- Site and mobile app shutdown
- A full **rebuild of the application**
- Engagement of security consultants (*NoMoreHacks Security Consultants*) for forensic analysis

Lesson Objective of the Case Study:
You are tasked with helping the company implement a strong **Security Development Lifecycle (SDL)** that includes:

- Cultural adoption of security practices
- Improved risk assessment
- SDL phases mapped into practical actions
- Drafting deliverables based on the 5 SDL phases (A1 through A5, plus PRSA1-5)

Study Tip: Review how each SDL phase (Ch. 3–8) would be applied to this company's situation. Think: What steps would you take if you were the security consultant?

✅ Appendix B: Answers to Chapter Quick-Check Questions

This appendix gives the **correct answers** to the review questions from each chapter. Examples include:

🔹 *Chapter 1 Sample Answers*

1. The cost to fix flaws post-release can be up to:
   ✅ **d. 1500 times** more than in development.
2. Defective software is:
   ✅ **d. A software development and engineering problem**
3. The three SDL goals:
   ✅ **c. Confidentiality, integrity, and availability (CIA Triad)**
4. Best time for threat modeling:
   ✅ **b. During product inception/product backlog development**

🔹 *Chapter 2 Sample Answers*

5. "Building Security In" begins with:
   ✅ **c. Specification phase**
6. SDL objectives **EXCLUDE**:
   ✅ **c. To eliminate threats to the software** (That's impossible)
7. Principle where entities have only necessary permissions:
   ✅ **b. Least privilege**
8. BSIMM is defined as:
   ✅ **c. Looking for evidence of security activities in the SDLC**

🧠 Chapter Quiz (with Answers)

11. **What is the first step in gaining executive buy-in for a software security program?**

a. Start code reviews
b. Build a pilot team
✅ **c. Show business value of security investment**
d. Implement encryption

12. **What role do "security champions" play?**
    a. Do all the security testing
    b. Build all the code
    ✅ **c. Advocate and model secure development practices**
    d. Write policy documents

13. **Which of the following is NOT a best practice for sustaining an SDL program?**
    a. Train developers
    b. Run periodic metrics
    ✅ **c. Replace the dev team every year**
    d. Use maturity models

14. **True or False: A security program should be entirely independent from the SDLC.**
    ✅ **False** — Integration is key for success.

15. **What is a key benefit of using a maturity model like BSIMM?**
    a. Avoid doing threat modeling
    ✅ **b. Measure and improve security practices over time**
    c. Replace developers
    d. Stop using secure code libraries

🧪 QUIZ QUESTIONS + ANSWERS

5. The following are primary mitigation methods **except**:

- ❌ a. Locking down the environment
- ❌ b. Input validation
- ✅ **c. Use of deprecated libraries for legacy code**
- ❌ d. Output validation

6. Elements of defensive coding include all **except**:

- ✅ **a. Custom cryptographic functions to avoid algorithm disclosure**
- ❌ b. Exception handling
- ❌ c. Interface coding
- ❌ d. Cryptographic agility

7. Static analysis checks for:

- ❌ a. Thread performance only
- ❌ b. Race conditions
- ✅ **c. Syntax, function calls, memory management**
- ❌ d. Logic semantics only

8. Automated testing **does not** have this advantage over manual code review:

- ❌ a. Unsafe function detection
- ❌ b. Obfuscated routines
- ❌ c. Speed
- ✅ **d. Integration into IDE**

Question 1

What is software security?

- Data transmission security by using HTTPS and SSL
- Security that websites use, such as Web Application Firewall to block and monitor HTTP traffic
- Security that networks use, such as a firewall allowing only intended traffic
- Security that deals with securing the foundational programmatic logic of the underlying software

Which part of the CIA goals keeps unauthorized users from accessing confidential information?

- Integrity
- Confidentiality
- Availability
- Information security

What are the three primary tools basic to the security development life cycle? Choose 3 answers.

- Fuzzing or fuzz testing
- Static analysis testing
- Dynamic analysis testing
- Software security architects
- Measurement model

"Fuzzing or fuzz testing" is correct. Fuzz testing is automated or semi-automated testing that provides invalid, unexpected, or random data to the computer software program. "Static analysis testing" is correct. Static analysis analyzes computer software without executing programs.

"*Dynamic analysis testing" is correct. Dynamic analysis analyzes computer software while executing programs.*

In which phase of the SDLC should the software security team be involved?

- Planning
- Support and Sustain
- Design and Development
- Release and Launch

- Concept

What determines the order of items in a product backlog in Scrum?

- Order is decided by the Scrum Team
- Order is decided by the ScrumMaster
- Order is decided by the project manager
- Order is decided based on value of the items being delivered

Order is decided based on the value of the item/requirement in the backlog as it helps business when the item is done and business can start using it. The Product Owner decides the order of items in the backlog.

Why is the Waterfall methodology most useful for smaller projects?

- When a project is smaller, it can easily be turned back upwards after the coding phase is complete.
- When a project is smaller, the risk of changing requirements and scope is lower.
- When a project is smaller, it doesn't need any time for reflection.
- When a project is smaller, there is an emphasis on empowering teams with collaborative decision-making.

The Waterfall method works with each stage being clearly defined. The project builds on itself, and in smaller projects, this creates a clearer and easily definable path.

✅ Chapter 1 Questions

1. **The costs to remediate security flaws once a software product is released can run as much as ____ times the cost to fix them in development:**

- a. 50
- b. 100
- c. **500**
- d. 1500

2. **Defective software is:**

- a. A network security problem
- b. An operating system security problem
- c. A user-caused problem
- **d. A software development and engineering problem**

3. **The three goals of the security development lifecycle are:**

- a. Reliability, efficiency, and maintainability
- b. Speed, quality, and continuous releases
- c. **Confidentiality, integrity, and availability**
- d. Availability, reliability, and portability

4. **Threat modeling and attack surface analysis is most effective when conducted:**

- a. Post-release
- b. During product inception/product backlog development
- c. During integration testing
- d. **Prior to code development/commitment**

✅ Chapter 2 Questions

5. **The paradigm of Building Security In begins with the:**

- a. Analysis phase
- b. Design phase
- c. Specification phase
- d. **Development phase**

6. **The objectives of SDL are to achieve all except:**

- a. Reduce the number of vulnerabilities
- b. Reduce severity of vulnerabilities
- c. Eliminate threats
- d. **Document a complete understanding of vulnerabilities**

7. **The principle that an object has only the necessary rights/privileges is:**

- a. Layered security
- b. **Least privilege**
- c. Role-based security
- d. Clark-Wilson model

8. **Which statement best describes BSIMM?**

- a. BSIMM is used to measure the maturity of a software assurance program by looking for evidence of security best practices in the SDLC
- b. … by looking for security procedures
- c. **… by looking for evidence of security activities in the SDLC**
- d. … by looking for security requirements

✅ **Chapter 3 Question**

9. **The purpose of the discovery meeting is to:**

- a. Discover who is on the development team
- b. Discover what budgets/resources are available
- c. **Discover how security can be built into the process from the start**
- d. Discover which platforms/languages will be used

What are the two common best principles of software applications in the development process? Choose 2 answers.

- **Quality code**
- **Secure code**
- Information security
- Integrity
- Availability

"Quality code" is correct. Quality code is efficient code that is easy to maintain and reusable.

"Secure code" is correct. Secure code authorizes and authenticates every user transaction, logs the transaction, and denies all unauthorized requisitions.

What ensures that the user has the appropriate role and privilege to view data?

- Authentication
- Multi-factor authentication
- Encryption
- Information security
- **Authorization**

Authorization ensures a user's information and credentials are approved by the system.

Which security goal is defined by "guarding against improper information modification or destruction and ensuring information non-repudiation and authenticity"?

- **Integrity**
- Quality
- Availability
- Reliability

The data must remain unchanged by unauthorized users and remain reliable from the data entry point to the database and back.

Which phase in an SDLC helps to define the problem and scope of any existing systems and determine the objectives of new systems?

- Requirements
- Design
- **Planning**
- Testing

The planning stage sets the project schedule and looks at the big picture.

What happens during a dynamic code review?

- **Programmers monitor system memory, functional behavior, response times, and overall performance.**
- Customers perform tests to check software meets requirements.
- An analysis of computer programs without executing them is performed.
- Input fields are supplied with unexpected input and tested.

This describes dynamic code review.

How should you store your application user credentials in your application database?

- Use application logic to encrypt credentials
- Store credentials as clear text
- Store credentials using Base 64 encoded
- **Store credentials using salted hashes**

Hashing is a one-way process that converts a password to ciphertext using hash algorithms. Password salting adds random characters before or after a password prior to hashing to obfuscate the actual password.

Which software methodology resembles an assembly-line approach?

- V-model
- Agile model
- Iterative model
- **Waterfall model**

Waterfall model is a continuous software development model in which the development steps flow steadily downwards.

Which software methodology approach provides faster time to market and higher business value?

- Iterative model
- Waterfall model
- V-model
- **Agile model**

In the agile model, projects are divided into small incremental builds that provide working software at the end of each iteration and adds value to business.

In Scrum methodology, who is responsible for making decisions on the requirements?

- Scrum Team
- **Product Owner**
- ScrumMaster
- Technical Lead

The Product Owner is responsible for requirements/backlog items and prioritizing them.

What is the product risk profile?

- A security assessment deliverable that lists education requirements for product and operations teams
- A security assessment deliverable that maps activities to the development schedule
- A security assessment deliverable that guides SDL activities to mitigate issues
- A security assessment deliverable that estimates the actual cost of the product

Looking at products from different perspectives allows management to determine the actual cost of a product, which includes selling it in different markets, and liabilities that might be incurred.

A software security team member has been tasked with creating a deliverable that provides details on where and to what degree sensitive customer information is collected, stored, or created within a new product offering.

What does the team member need to deliver in order to meet the objective?

- Threat profile
- **Privacy impact assessment**
- Metrics template
- SDL project plan

The PIA is a process that evaluates issues and privacy impact rating in relation to the privacy of personally identifiable information in the software.

- **A software security team member has been tasked with creating a threat model for the login process of a new product.**
  What is the first step the team member should take?
- Identify threats
- Survey the application
- Decompose the application
- **Identify security objectives**

What are three parts of the STRIDE methodology?
Choose 3 answers.

- **Spoofing**
- **Elevation**
- **Tampering**
- Trike
- Threat source
- Vulnerability

Why:
STRIDE is a threat modeling framework developed by Microsoft, with six core categories:

| ***S*** | ***Spoofing identity***
| ***T*** | ***Tampering with data***
| ***R*** | ***Repudiation***
| ***I*** | ***Information Disclosure***
| ***D*** | ***Denial of Service (DoS)***
| ***E*** | ***Elevation of Privilege***

Trike, Vulnerability, *and* ***Threat Source*** *are not STRIDE categories.*

What is the reason software security teams host discovery meetings with stakeholders early in the development life cycle?

- To determine how much budget is available for new security tools
- To meet the development team
- To refactor functional requirements to ensure security is included
- **To ensure that security is built into the product from the start**

To correctly and cost-effectively introduce security into the software development life cycle, it needs to be done early.

Why should a security team provide documented certification requirements during the software assessment phase?

- Certification is required if the organization wants to move to the cloud.
- **Depending on the environment in which the product resides, certifications may be required by corporate or government entities before the software can be released to customers.**
- By ensuring software products are certified, the organization is protected from future litigation.
- By ensuring all developers have security certifications before writing any code, teams can forego discovery sessions.

Any new product may need to be certified based on the data it stores, the frameworks it uses, or the domain in which it resides. Those certification requirements need to be analyzed and documented early in the development life cycle.

What are two items that should be included in the privacy impact assessment plan regardless of which methodology is used?
Choose 2 answers.

- **Required process steps**
- **Technologies and techniques**
- SDL project outline
- Threat modeling
- Post-implementation signoffs

"Required process steps" is correct. Required process steps explain in more detail which requirements are relevant to developers, detailing what types of data are considered sensitive and how they need to be protected.

What are the goals of each SDL deliverable?
Product risk profile

Answer: Estimate the actual cost of the product

SDL Project Outline?
Answer: Map Security Activities to the development schedule

Threat Profile?

Answer: Guide Security activities to protect the product from vulnerabilities

List third-party software?

Answer: Identify dependence on unmanaged software

What is a threat action that is designed to illegally access and use another person's credentials?

- Tampering
- **Spoofing**
- Elevation of privilege
- Information disclosure

Spoofing is a threat action that occurs when the cyber criminal acts as a trusted device to get you to relay secure information.

What are two steps of the threat modeling process?
Choose 2 answers.

- **Survey the application**
- **Decompose the application**
- Redesign the process to eliminate the threat
- Transfer the risk
- Identify business requirements

"Survey the application" is correct. Surveying the application is a way to gain knowledge of how the product works by reading product documentation and interviewing the development team.

"Decompose the application" is correct. Decomposing the application can be done by doing a deep dive into the code and understanding how it works behind the scenes.

Which shape indicates each type of flow diagram element?

External elements:

Answer: Rectangle

Data Store?

Answer: Two parallel horizontal lines

Data Flow?

Answer: Solid Line with an arrow

Trust Boundary:

Answer: DASHED LINE

What are the two deliverables of the Architecture phase of the SDL?
Choose 2 answers.

- **Threat modeling artifacts**
- **Policy compliance analysis**
- Information disclosure
- Attack modeling
- Application decomposition

"Threat modeling artifacts" is correct. Threat modeling artifacts include data flow diagrams, technical threat modeling reports, high-level executive threat modeling reports, and recommendations for threat analysis.
"Policy compliance analysis" is correct. Policy compliance analysis is a report on compliance with security and non-security policies of the organization.

What SDL security assessment deliverable is used as an input to an SDL architecture process?

- SDL project outline
- Certification requirements
- Product risk profile
- **Threat profile**

Threat profiles created in the Security Assessment phase are used to build the environment in which the product will operate and will include potential threats in order to determine how to avoid them in the final application.

Which software security testing technique tests the software from an external perspective?

- Source code analysis
- White box
- Gray box
- Black box

Black box testing tests with no prior knowledge of the software. During this phase, only binary executable or intermediate byte code is analyzed.

Which security design principle states that an entity should be given the minimum privileges and resources for a minimum period of time for a task?

- Defense in depth
- Least privilege
- Economy of mechanism
- Separation of duties

By providing the least amount of privilege, opportunities for unauthorized access to sensitive information are eliminated.

LESSON 7 QUIZ

After the developer is done coding a functionality, when should code review be completed?

✔ ***Correct Answer: Within hours or the same day***

*Code reviews are most effective when done **immediately after coding**, to catch issues early and reduce context switching.*

What is the order that code reviews should follow in order to be effective?

Step 1: Plan the review

Identify security Code Review objectives

Step 2: Perform the review

Perform a Preliminary scan

Step 3: Record defects

Review code for security issues

Step 4: Fix defects and verify

Review for security issues unique to the architecture

✅ **D487 Code Review Process Table**

Step	Phase	Description
1	Identify Security Code Review Objectives	Define scope, goals, and constraints. Focus on specific threats (e.g., auth, input validation).
2	Perform Preliminary Scan	Run static analysis tools to detect obvious or known security flaws automatically.
3	Review Code for Security Issues	Manually inspect code for vulnerabilities (e.g., XSS, SQLi, insecure crypto, logic errors).
4	Review for Architecture-Specific Issues	Analyze for flaws unique to the app's architecture (e.g., broken trust boundaries, insecure design).

✅ D487 Quiz – Questions and Correct Answers

Question 1:
When a software application handles personally identifiable information (PII) data, what will be the Privacy Impact Rating?
✔ **Answer:** P1: High privacy risk

Question 2:
Which key success factor identifies threats to the software?
✔ **Answer:** Effective threat modeling

Question 3:
What is the goal of design security review deliverables?
✔ **Answer:** To make modifications to the design of software components based on security assessments

---

Question 4:
Which application scanner component is useful in identifying vulnerabilities such as cookie misconfigurations and insecure configuration of HTTP response headers?
✔ **Answer:** Passive scanner

Question 5:
Which type of attack occurs when an attacker uses malicious code in the data sent in a form?
✔ **Answer:** Cross-site scripting

Question 6:
Which tools provide the given functions?

6a: Self-managed, automatic code review product
✔ **Answer:** SonarQube

6b: Open-source automation server
✔ **Answer:** Jenkins

6c: Proprietary issue tracking product

✔ **Answer:** JIRA

6d: AI-powered management solution

✔ **Answer:** Dynatrace

Question 7:

A new application is released, and users perform initial testing on the application.
Which type of testing are the users performing?

✔ **Answer:** Beta testing

Question 8:

What is a non-system-related component in software security testing attack surface validation?

✔ **Answer:** Users

Question 9:

When an application's input validation is not handled properly, it could result in which kind of vulnerabilities?

✔ **Answer:** SQL injection, cross-site scripting

Question 10:

What are the advantages of the following security analysis tools?

10a: Static code analysis

✔ **Answer:** Access to the actual instructions the software will be guessing

10b: Dynamic code analysis

✔ **Answer:** Tests a specific operational deployment

10c: Fuzz testing

✔ **Answer:** Testing in a random approach

10d: Manual source code review

✔ **Answer:** Requires no supporting technology

Which activity in the Ship (A5) phase of the security development cycle sets requirements for quality gates that must be met before release?

✅ **Answer:** A5 policy compliance analysis

🦣 This step ensures **security policy alignment and defines required quality/security gates** before release approval.

The company's website uses querystring parameters to filter products by category. The URL, when filtering on a product category, looks like this: company.com/products?category=2.

If the security team saw a URL of company.com/products?category=2 OR 1=1 in the logs, what assumption should they make?

✅ **Answer:** An attacker is attempting to use SQL injection to gain access to information.

🦣 *This is a classic **SQL Injection** pattern. OR 1=1 always returns true, potentially exposing data.*

Which post-release support activity (PRSA) details the process for investigating, mitigating, and communicating findings when security vulnerabilities are discovered in a software product?

✅ **Correct Answer:** External vulnerability disclosure response

🦣 *This activity defines how vulnerabilities discovered **after release** are tracked, investigated, mitigated, and disclosed to stakeholders or customers.*

Which post-release support key success factor says that any change or component reuse should trigger security development life cycle activities?

✅ **Correct Answer:** SDL cycle for any architectural changes or code reuses

🦣 *This ensures that **any new deployment, reuse, or architectural change** (e.g., moving to cloud, combining systems) triggers **a fresh round of SDL tasks** (threat modeling, review, etc.).*

Which step will you find in the SANS Institute Cyber Defense seven-step recipe for conducting threat modeling and application risk analysis?

✔ **Answer:** Brainstorm threats from adversaries

🐘 This is part of the SANS model. It emphasizes identifying potential adversaries and how they might exploit the system.

In which OpenSAMM core practice area would one find environment hardening?

✔ **Answer:** Deployment

🐘 Environment hardening (e.g., server config, firewalls, patching) is part of the **Deployment** phase in OpenSAMM.

CHAPTER 10 QUIZ

1/ Which practice in the Ship (A5) phase of the security development cycle verifies whether the product meets security mandates?

✔ **Correct Answer:** A5 policy compliance analysis

📘 *Explanation: This ensures products meet internal and external requirements, including quality gates and SDL activities.*

2/ Which post-release support activity defines the process to communicate, identify, and alleviate security threats?

✔ **Correct Answer:** PRSA1: External vulnerability disclosure response

📘 *Explanation: This step defines how an organization discloses, evaluates, mitigates, and communicates post-release vulnerabilities*.

3/ What are two core practice areas of the OWASP Security Assurance Maturity Model (OpenSAMM)?

✔ **Correct Answers:**

- Governance
- Construction

📘 *Explanation: Governance manages policies and organizational practices. Construction deals with secure coding and dev practices.*

4/ Which practice in the Ship (A5) phase of the security development cycle uses tools to identify weaknesses in the product?

✔ **Correct Answer:** Vulnerability scan

📘 *Explanation: Automated tools detect known weaknesses based on signature databases.*

5/ Which post-release support activity should be completed when companies are joining together?

✔ **Correct Answer:** Internal review

📘 *Explanation: Internal reviews validate security and architecture during integrations or mergers.*

6/ Match each Ship (A5) activity with its action

6a. A5 Policy compliance analysis

✔ **Answer:** Analyze activities and standards

📘 *Explanation: Ensures SDL activities and quality gates were completed at each phase.*

6b. Code-assisted penetration testing

✔ **Answer:** White-box security test

📘 *Explanation: Simulates a hacker using knowledge of the system to find vulnerabilities.*

6c. Open-source licensing review

✔ **Answer:** License compliance

📘 *Explanation: Confirms all OSS components are compliant and won't delay release.*

6d. Final security review

✔ **Answer:** Release and ship

📘 *Explanation:* Final testing (including regression tests) confirms security readiness for shipping.

7/ Match SDL implementation by environment

7a. Agile

✅ **Answer:** Iterative development

📘 *Explanation: Agile uses short development cycles and evolves through collaboration.*

7b. DevOps

✅ **Answer:** Continuous integration and continuous deployment

📘 *Explanation: DevOps emphasizes automation and real-time feedback throughout development and delivery.*

7c. Cloud

✅ **Answer:** API invocation processes

📘 *Explanation: Cloud applications rely on dynamic APIs and data exchange mechanisms.*

7d. Digital enterprise

✅ **Answer:** Enables and improves business activities

📘 *Explanation: Focuses on digitizing business processes to drive performance and scalability.*

Question 8

Which phase of penetration testing allows for remediation to be performed?

✅ **Correct Answer:** Assess

📘 *Explanation:* The Assess phase executes the test and allows time for fixing discovered issues.

Question 9

Which key deliverable occurs during post-release support?

✅ **Correct Answer:** Third-party reviews

📘 *Explanation:* Security audits from external groups validate the product post-launch.

10/ Match OpenSAMM core practices to business functions

10a. Governance

✅ **Answer:** Policy and compliance

📘 *Explanation: Manages security policies, controls, and audits.*

10b. Construction

✅ **Answer:** Threat assessment

📘 *Explanation: Involves identifying and characterizing threats in the codebase.*

10c. Verification

✅ **Answer:** Code review

📘 *Explanation: Manual or tool-based review of source code to detect flaws.*

10d. Deployment

✅ **Answer:** Vulnerability management

📘 *Explanation: Involves handling both internal and external reports of vulnerabilities.*