Complete 26-Week Application Security Engineering Curriculum

Target: Remote AppSec Engineer (25K-45K)

Complete 26-Week Application Security Engineering Curriculum

Target Role: Remote AppSec Engineer at Trail of Bits, NCC Group, Anthropic, GitLab, Stripe, or Coinbase **Salary Range:** $125K-$145K **Timeline:** December 2, 2025 – June 1, 2026 **Total Hours:** 572 hours (22 hours/week average)

Phase 1: Foundation (Weeks 1-3, 66 hours)

Week 1: Python Fundamentals & Web Security Basics (22 hours)

Python Workout Ch 1-2 (4 hours): Number guessing game, summing numbers, running average, hexadecimal output, mysum function, run timing, word statistics

Effective Python Items 1-5 (2 hours - OPTIONAL): Know which Python version you're using, follow PEP 8 style guide, bytes vs str differences, f-strings over format strings, helper functions over complex expressions

Security Study (8 hours) - EXPLICIT READINGS:

1. **OWASP Top 10 2021 - Complete Overview** (2 hours) • URL: https://owasp.org/Top10/ • Focus: A01 Broken Access Control, A02 Cryptographic Failures, A03 Injection • Study all 10 categories with real-world examples • Map vulnerabilities to CWE entries
2. **PortSwigger Web Security Academy - SQL Injection** (3 hours) • URL: https://portswigger.net/web-security/sql-injection • Focus: UNION attacks, blind SQLi, second-order injection • Complete reading materials before labs
3. **OWASP Testing Guide v4.2 - Section 4.7 Input Validation** (2 hours) • URL: https://owasp.org/www-project-web-security-testing-guide/v42/ • Focus: WSTG-INPV-05 SQL Injection testing methodology • Document testing procedures
4. **OWASP Cheat Sheet - SQL Injection Prevention** (1 hour) • URL: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html • Focus: Parameterized queries, stored procedures, allowlist validation

Practice (4 hours): Build SQL injection detection script with Python, implements pattern matching for common SQLi payloads (UNION, OR 1=1, comment injection), outputs findings in JSON format with severity ratings

Labs (4 hours): PortSwigger SQL injection (8 labs: 4 Apprentice + 4 Practitioner, total: 8/211)

Deliverable: SQLi detection script, 8 labs complete, OWASP Top 10 summary notes

---

Week 2: Functions, Decorators & XSS Fundamentals (22 hours)

Python Workout Ch 3-4 (4 hours): First-last function, myxml element generator, copyfile, password generator, factorial, prefix notation calculator

Effective Python Items 6-10 (2 hours - OPTIONAL): Multiple assignment unpacking, prefer enumerate over range, use zip for parallel iteration, avoid else after loops, assignment expressions

Security Study (8 hours) - EXPLICIT READINGS:

1. **PortSwigger - Cross-Site Scripting (XSS) Complete Guide** (3 hours) • URL: https://portswigger.net/web-security/cross-site-scripting • Focus: Reflected, Stored, and DOM-based XSS differences • Study contexts: HTML, JavaScript, attribute injection
2. **OWASP Cheat Sheet - XSS Prevention** (1.5 hours) • URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html • Focus: Output encoding rules, context-specific escaping • Document encoding functions for each context
3. **OWASP Cheat Sheet - DOM-based XSS Prevention** (1.5 hours) • URL: https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html • Focus: Dangerous sinks, safe assignment methods • Create reference card for DOM manipulation
4. **PortSwigger - Content Security Policy (CSP)** (2 hours) • URL: https://portswigger.net/web-security/cross-site-scripting/content-security-policy • Focus: CSP directives, bypass techniques, nonce implementation • Document CSP header configurations

Practice (4 hours): Build XSS payload generator with modular functions for different XSS types (reflected, stored, DOM), uses decorators for payload encoding (HTML entities, URL encoding, JavaScript unicode escaping), implements context-aware payload selection

Labs (4 hours): PortSwigger XSS labs (10 labs: 5 Apprentice + 5 Practitioner, total: 18/211)

Deliverable: XSS payload generator with encoding decorators, 18 labs complete, XSS context cheat sheet

---

Week 3: Files, Data Processing & Authentication Attacks (22 hours)

Python Workout Ch 5 (3 hours): Final line, /etc/passwd to dict, word count, longest word per file, reading and writing CSV, JSON parsing, reverse lines

Effective Python Items 11-15 (2 hours - OPTIONAL): Sequence slicing, avoid striding with slicing, catch-all unpacking, sort by complex criteria with key parameter, dict insertion ordering

Security Study (8 hours) - EXPLICIT READINGS:

1. **PortSwigger - Authentication Vulnerabilities** (3 hours) • URL: https://portswigger.net/web-security/authentication • Focus: Password-based attacks, multi-factor bypass, brute force protection • Study username enumeration techniques
2. **API Security in Action - Chapter 3: Securing the Natter API** (2 hours) • Focus: HTTP Basic authentication, secure password storage with Scrypt • Study HTTPS/TLS configuration, audit logging setup • Pages: ~60-90
3. **OWASP Cheat Sheet - Authentication** (1.5 hours) • URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html • Focus: Credential storage, session management integration • Document secure authentication patterns
4. **OWASP Cheat Sheet - Password Storage** (1.5 hours) • URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html • Focus: Argon2id, bcrypt, scrypt comparison • Create algorithm selection decision tree

Practice (5 hours): Build credential analysis tool that parses authentication logs (CSV/JSON), identifies brute force patterns using timing analysis, generates security reports with risk scores, implements decorator for timing attack detection

Labs (4 hours): PortSwigger Authentication labs (8 labs: 4 Apprentice + 4 Practitioner, total: 26/211)

Deliverable: Credential analysis tool, authentication pattern report, 26 labs complete

---

Phase 2: Exploitation (Weeks 4-6, 66 hours)

Week 4: Functions Deep Dive & Access Control (22 hours)

Python Workout Ch 6 (3 hours): XML generator function, prefix notation calculator, password generator with configurable complexity

Effective Python Items 16-21 (2 hours - OPTIONAL): Prefer get for missing dict keys, defaultdict over setdefault, **missing** for key-dependent defaults, limit unpacked variables, raise exceptions over returning None, closures and variable scope

Security Study (8 hours) - EXPLICIT READINGS:

1. **PortSwigger - Access Control Vulnerabilities** (3 hours) • URL: https://portswigger.net/web-security/access-control • Focus: Vertical privilege escalation, IDOR, horizontal privilege escalation • Study parameter-based access control flaws
2. **Hacking APIs - Chapter 10: Exploiting Authorization** (2.5 hours) • Focus: BOLA discovery, resource ID analysis, A-B testing methodology • Study BFLA (Broken Function Level Authorization) patterns • Pages: ~200-230
3. **OWASP API Security Top 10 - API1:2023 BOLA** (1.5 hours) • URL: https://owasp.org/API-Security/editions/2023/en/0x11-t10/ • Focus: Broken Object Level Authorization patterns • Map to real-world API vulnerability examples
4. **OWASP Cheat Sheet - Authorization** (1 hour) • URL: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html • Focus: RBAC vs ABAC, permission inheritance models

Practice (5 hours): Build IDOR scanner that fuzzes object identifiers (sequential, UUID, encoded), implements A-B testing logic for authorization bypass, uses generator functions for payload creation, outputs vulnerability report with reproduction steps

Labs (4 hours): PortSwigger Access Control labs (10 labs: 5 Apprentice + 5 Practitioner, total: 36/211)

Deliverable: IDOR scanner tool, access control testing methodology document, 36 labs complete

---

Week 5: List Comprehensions & CSRF/SSRF (22 hours)

Python Workout Ch 7 (3 hours): Join numbers, add numbers from nested structures, flatten a list, Pig Latin translation of a file, flip a dict, transform values, supervocalic words, gematria parts 1 & 2

Effective Python Items 22-26 (2 hours - OPTIONAL): Variable positional arguments, keyword arguments for optional behavior, None and docstrings for dynamic defaults, keyword-only and positional-only arguments, functools.wraps for decorators

Security Study (8 hours) - EXPLICIT READINGS:

1. **PortSwigger - CSRF Complete Guide** (2.5 hours) • URL: https://portswigger.net/web-security/csrf • Focus: Token bypass techniques, SameSite cookie bypass, CORS misconfigurations • Study referer header validation flaws
2. **PortSwigger - SSRF Complete Guide** (2.5 hours) • URL: https://portswigger.net/web-security/ssrf • Focus: Basic SSRF, blind SSRF, SSRF with whitelist bypass • Study cloud metadata exploitation (169.254.169.254)
3. **API Security in Action - Chapter 10: Microservice APIs** (2 hours) • Focus: SSRF attacks in microservices, DNS rebinding attacks • Study network policy configuration • Pages: ~280-310
4. **OWASP Cheat Sheet - SSRF Prevention** (1 hour) • URL: https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html • Focus: Allowlist validation, network segmentation

Practice (5 hours): Build CSRF PoC generator using list comprehensions for form field extraction, generates HTML forms with hidden fields, implements SameSite bypass payloads, creates clickjacking iframe payloads

Labs (4 hours): PortSwigger CSRF (6 labs) + SSRF (5 labs, total: 47/211)

Deliverable: CSRF PoC generator, SSRF payload collection, 47 labs complete

---

Week 6: Review & Foundation Assessment (22 hours)

Python Review (3 hours): Review chapters 1-7 exercises, refactor previous tools using comprehensions and decorators

Effective Python Items 27-30 (2 hours - OPTIONAL): Comprehensions over map/filter, limit comprehension subexpressions, assignment expressions in comprehensions, generators over returning lists

Security Study (6 hours) - EXPLICIT READINGS:

1. **OWASP ASVS v5.0 - Chapters 1-3** (3 hours) • URL: https://github.com/OWASP/ASVS • Focus: Verification levels, architecture requirements • Map your tools to ASVS requirements
2. **Full Stack Python Security - Chapter 1: Defense in Depth** (1.5 hours) • Focus: Layered security model, security as a continuous process • Pages: ~1-25
3. **Hacking APIs - Chapter 3: Common API Vulnerabilities** (1.5 hours) • Focus: Information disclosure, excessive data exposure • Create API vulnerability checklist • Pages: ~50-80

Practice (5 hours): Consolidate and refactor all tools from Weeks 1-5 into unified security toolkit, implement consistent error handling, add logging with security context, create CLI interface with argparse

Labs (4 hours): Complete remaining Apprentice-level labs from any category (8 labs, total: 55/211)

Writing (2 hours): Blog post: “Building Your First Security Testing Toolkit in Python: Lessons from 55 Labs”

Deliverable: Unified security toolkit v1.0, blog post draft, 55 labs complete, foundation skills mastery

---

Phase 3: Authentication & Authorization (Weeks 7-9, 66 hours)

Week 7: Modules/Packages & OAuth 2.0 Deep Dive (22 hours)

Python Workout Ch 8 (3 hours): Sales tax module with regional rates, menu package system with submodules, module organization patterns

Effective Python Items 31-36 (2 hours - OPTIONAL): Be defensive iterating over arguments, generator expressions for large comprehensions, yield from for composition, avoid send in generators, avoid throw in generators, itertools for iterators

Security Study (8 hours) - EXPLICIT READINGS:

1. **API Security in Action - Chapters 6-7: JWTs and OAuth2** (4 hours) • Chapter 6 Focus: JWT claims, JOSE header, generating/validating JWTs, encrypted JWTs • Chapter 7 Focus: OAuth2 flows, Authorization Code grant, PKCE, refresh tokens, OpenID Connect • Pages: ~150-220
2. **PortSwigger - OAuth Authentication Vulnerabilities** (2 hours) • URL: https://portswigger.net/web-security/oauth • Focus: Authorization code injection, token leakage, PKCE bypass
3. **Full Stack Python Security - Chapter 11: OAuth 2** (2 hours) • Focus: Django OAuth implementation, third-party access patterns • Pages: ~220-250

Practice (5 hours): Build OAuth flow analyzer as Python package, implements token inspection (JWT decode, signature validation), detects common misconfigurations (missing state, no PKCE), generates security assessment report

Labs (4 hours): PortSwigger OAuth labs (6 labs) + JWT labs (4 labs, total: 65/211)

Deliverable: OAuth analyzer package, JWT decoder module, 65 labs complete

---

Week 8: OOP Part 1 & Complete Authentication System (22 hours)

Python Workout Ch 9 (3 hours): Ice cream scoop class, ice cream bowl class, bowl limits implementation, bigger bowl with inheritance

Effective Python Items 37-41 (2 hours - OPTIONAL): Compose classes over nesting built-in types, functions for simple interfaces, @classmethod polymorphism, initialize parents with super, mix-in classes for functionality

Security Study (8 hours) - EXPLICIT READINGS:

1. **Full Stack Python Security - Chapters 7-9: Sessions, Authentication, Passwords** (4 hours) • Chapter 7: HTTP session management, session cookies, Django sessions • Chapter 8: User authentication patterns, login systems • Chapter 9: Password storage, salting, secure policies • Pages: ~140-200
2. **API Security in Action - Chapter 4: Session Cookie Authentication** (2 hours) • Focus: Token-based auth, session fixation attacks, CSRF prevention, SameSite cookies • Pages: ~90-120
3. **Hacking APIs - Chapter 8: Attacking Authentication** (2 hours) • Focus: Password brute-force, token forging, JWT None attack, algorithm confusion • Pages: ~160-190

Practice (5 hours): Build secure authentication module using OOP: User class with password hashing (Argon2id), Session class with secure token generation, AuthManager class coordinating user/session lifecycle, implements rate limiting decorator

Labs (4 hours): PortSwigger Authentication advanced (6 labs) + Business Logic (4 labs, total: 75/211)

Deliverable: Authentication module with OOP patterns, secure session implementation, 75 labs complete

---

Week 9: OOP Part 2 & P2P Exercise Creation (22 hours)

Python Workout Ch 9 continued (3 hours): FlexibleDict class, animals with inheritance hierarchy, cages class composition, zoo management system

Effective Python Items 42-46 (2 hours - OPTIONAL): Prefer public attributes over private, inherit from collections.abc, plain attributes over setters/getters, @property over refactoring, descriptors for reusable @property

Security Study (6 hours) - EXPLICIT READINGS:

1. **API Security in Action - Chapters 8-9: Identity-based Access Control** (3 hours) • Chapter 8: RBAC, ABAC, policy agents, XACML • Chapter 9: Capability-based security, macaroons with caveats • Pages: ~220-280

2. **Full Stack Python Security - Chapter 10: Authorization** (2 hours) • Focus: Permission systems, role-based access in Django • Pages: ~200-220

3. **OWASP Cheat Sheet - Access Control** (1 hour) • URL: https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html • Focus: Enforce access control on every request, deny by default

Practice (4 hours): Build RBAC/ABAC demonstration framework using OOP, Policy class hierarchy with enforcement methods, implements permission checking decorators, creates audit trail with observer pattern

P2P Project (5 hours): Create 5 P2P authentication/authorization exercises with 30+ tests each: • Exercise 1: Implement secure password hashing with timing-safe comparison • Exercise 2: Build JWT token generator/validator with proper claims • Exercise 3: Create session management with secure cookie attributes • Exercise 4: Implement RBAC permission checker • Exercise 5: Build OAuth authorization code flow validator

Writing (2 hours): Blog post: “5 Authorization Bugs AI Code Generation Consistently Misses”

Deliverable: RBAC framework, 5 P2P exercises with 150+ tests, blog post, authentication phase complete

---

Phase 4: Security Automation (Weeks 10-12, 66 hours)

Week 10: Iterators/Generators & SAST Tools (22 hours)

Python Workout Ch 10 (3 hours): MyEnumerate implementation, circle iterator, all lines all files generator, elapsed since iterator, MyChain implementation

Effective Python Items 47-51 (2 hours - OPTIONAL): getattr and **getattribute** for lazy attributes, validate subclasses with **init_subclass**, register classes with **init_subclass**, **set_name** for class attributes, class decorators over metaclasses

Security Study (8 hours) - EXPLICIT READINGS:

1. **Semgrep Official Documentation - Getting Started & Custom Rules** (3 hours) • URL: https://semgrep.dev/docs/ • Focus: Rule syntax, pattern matching, metavariables • Practice: Create 5 custom rules for Python security issues
2. **Bandit Documentation - Python Security Analysis** (2 hours) • URL: https://bandit.readthedocs.io/ • Focus: Built-in plugins, severity levels, baseline configuration • Study: AST-based detection methodology
3. **OWASP Code Review Guide - SAST Concepts** (1.5 hours) • URL: https://owasp.org/www-project-code-review-guide/ • Focus: Taint analysis, data flow tracking
4. **SonarQube Community Edition Documentation** (1.5 hours) • URL: https://docs.sonarsource.com/sonarqube-community-build/ • Focus: Quality gates, security rules, Python plugin

Practice (5 hours): Build SAST orchestrator using generators: iterates through codebase files yielding findings, implements parallel scanning with yield from, creates custom Semgrep rule generator, aggregates results from multiple tools (Bandit, Semgrep)

Labs (4 hours): OWASP Juice Shop challenges (15 challenges: injection, XSS, authentication) • URL: https://owasp.org/www-project-juice-shop/

Deliverable: SAST orchestrator with generator patterns, 5 custom Semgrep rules, Juice Shop progress

---

Week 11: Async Foundations & CI/CD Security (22 hours)

Python Workout Review (2 hours): Review all chapters, identify patterns applicable to async programming

Effective Python Items 52-57 (3 hours - OPTIONAL): subprocess for child processes, threads for blocking I/O, Lock for thread safety, Queue for thread coordination, recognize when concurrency is necessary, avoid creating threads for fan-out

Security Study (8 hours) - EXPLICIT READINGS:

1. **GitHub Actions Security Hardening Guide** (3 hours) • URL: https://docs.github.com/en/actions/security-for-github-actions/security-guides/security-hardening-for-github-actions • Focus: GITHUB_TOKEN permissions, secrets management, self-hosted runner risks • Create: Secure workflow template library
2. **GitLab CI/CD Security Documentation** (2 hours) • URL: https://docs.gitlab.com/ci/pipeline_security/ • Focus: Protected variables, job tokens, security scanning integration
3. **Pre-commit Framework Documentation** (1.5 hours) • URL: https://pre-commit.com/ • Focus: Hook configuration, security tool integration • Setup: Security-focused pre-commit config
4. **OWASP DevSecOps Guideline** (1.5 hours) • URL: https://owasp.org/www-project-devsecops-guideline/ • Focus: Pipeline security gates, shift-left security

Practice (5 hours): Build CI/CD security scanner: analyzes GitHub Actions/GitLab CI YAML files, detects insecure patterns (hardcoded secrets, overly permissive tokens, unpinned actions), generates remediation report with secure alternatives

Labs (4 hours): Implement security scanning in sample CI/CD pipelines: • Setup GitHub Actions workflow with Semgrep, Bandit, Gitleaks • Configure pre-commit hooks for local security checks • Create pull request security gate

Deliverable: CI/CD YAML analyzer, secure workflow templates, pre-commit security config

---

Week 12: Review & Security Framework Dashboard (22 hours)

Python Review (2 hours): Consolidate all OOP and generator patterns from Weeks 7-11

Effective Python Items 58-64 (3 hours - OPTIONAL): Queue refactoring for concurrency, ThreadPoolExecutor, coroutines for concurrent I/O, port threaded I/O to asyncio, mix threads and

coroutines, avoid blocking event loop, concurrent.futures for parallelism

Security Study (6 hours) - EXPLICIT READINGS:

1. **OWASP ZAP Documentation - API Scanning** (2.5 hours) • URL: https://www.zaproxy.org/docs/docker/api-scan/ • Focus: Automation modes, OpenAPI import, CI/CD integration • Practice: Script ZAP scans with Python API
2. **Nuclei Template Documentation** (2 hours) • URL: https://docs.projectdiscovery.io/opensource/nuclei/overview • Focus: Template syntax, severity classification, workflow chaining
3. **Hacking APIs - Chapter 4: Your API Hacking System** (1.5 hours) • Focus: Tool ecosystem integration, Postman automation • Pages: ~80-110

Practice (7 hours): Build security framework dashboard integrating: • SAST results aggregation (Semgrep, Bandit findings) • DAST integration (ZAP API for on-demand scanning) • Dependency vulnerability display (Safety/pip-audit output) • Secret scanning status (Gitleaks results) • Generate HTML dashboard with risk scores and trends • Use OOP for tool abstraction and generators for result streaming

Writing (2 hours): Blog post: “Building a Python Security Dashboard: Integrating 5 Tools in 200 Lines”

P2P Project (2 hours): Create P2P exercises for SAST/CI integration with automated grading

Deliverable: Security dashboard v1.0, blog post, automation skills mastery

---

Phase 5: Research & Discovery (Weeks 13-14, 44 hours)

Week 13: Advanced Iteration & API Security Scanner (22 hours)

Python Advanced (3 hours): Review itertools module, implement custom security-focused iterators for fuzzing patterns

Effective Python Items 65-70 (2 hours - OPTIONAL): try/except/else/finally blocks, contextlib and with statements, datetime over time for local clocks, pickle reliability with copyreg, decimal for precision, profile before optimizing

Security Study (8 hours) - EXPLICIT READINGS:

1. **Hacking APIs - Chapters 5-7: Discovery & Analysis** (4 hours) • Chapter 5: Setting up vulnerable API targets (crAPI, DVGA) • Chapter 6: Passive/active reconnaissance, Kiterunner, API discovery • Chapter 7: Endpoint analysis, reverse engineering APIs • Pages: ~110-160
2. **OWASP API Security Top 10 - Full Review** (2 hours) • URL: https://owasp.org/API-Security/editions/2023/en/0x11-t10/ • Focus: API1-API10 with detection methodologies • Create: API security testing checklist
3. **PortSwigger - API Testing Guide** (2 hours) • URL: https://portswigger.net/web-security/api-testing • Focus: API documentation exploitation, hidden endpoints

Practice (5 hours): Build API security scanner with itertools: • Endpoint discovery using itertools.product for path permutations • Parameter fuzzing with itertools.chain and cycle • Authentication bypass testing with permutation patterns • Rate limit detection and evasion • JSON/XML parser for response analysis

Labs (4 hours): PortSwigger API Testing labs (5 labs) + Business Logic (5 labs, total: 85/211)

Deliverable: API security scanner v1.0, API testing checklist, 85 labs complete

---

Week 14: Review & PyJWT Security Audit (22 hours)

Python Review (2 hours): Consolidate async patterns, prepare for enterprise tooling

Effective Python Items 71-76 (2 hours - OPTIONAL): deque for producer-consumer queues, bisect for sorted sequences, heapq for priority queues, memoryview and bytearray for zero-copy

Security Study (6 hours) - EXPLICIT READINGS:

1. **Hacking APIs - Chapters 13-14: Evasion & GraphQL** (3 hours) • Chapter 13: Security control detection, rate limit bypass, IP rotation • Chapter 14: GraphQL requests, introspection, InQL Burp extension • Pages: ~280-330
2. **PortSwigger - JWT Attacks Complete Guide** (2 hours) • URL: https://portswigger.net/web-security/jwt • Focus: Algorithm confusion, key injection, JKU/X5U attacks
3. **Critical JWT Vulnerabilities Research** (1 hour) • Study CVE-2015-9235 (None algorithm) • Study CVE-2016-10555 (Algorithm confusion) • Review PyJWT changelog for security fixes

Practice (8 hours): Conduct PyJWT security audit: • Clone PyJWT repository from GitHub • Review cryptographic implementation • Test for algorithm confusion vulnerabilities • Check default configurations for security issues • Document findings in security audit report format • Submit responsible disclosure if issues found

Writing (2 hours): Blog post: “What I Learned Auditing PyJWT: A Junior Security Engineer’s Perspective”

P2P Project (2 hours): Create JWT security exercise set with vulnerable implementations to fix

Deliverable: PyJWT audit report, JWT attack tool, blog post, 85 labs maintained

---

Phase 6: Enterprise Production Tools (Weeks 15-17, 66 hours)

Week 15: Enterprise Security Framework Design (22 hours)

Python Advanced (3 hours): Design patterns for enterprise applications: Factory, Strategy, Observer for security tools

Effective Python Items 77-81 (2 hours - OPTIONAL): repr strings for debugging, TestCase subclasses for related behaviors, setUp/tearDown for test isolation, mocks for complex dependencies, encapsulate dependencies for testing

Security Study (8 hours) - EXPLICIT READINGS:

1. **Secure by Design - Chapters 7-8: State & Pipeline Security** (4 hours) • Chapter 7: Partially immutable entities, entity state objects, snapshots, entity relay • Chapter 8: Security in delivery pipeline, unit tests for security, feature toggle security, automated security tests • Focus: Deep modeling for security
2. **Full Stack Python Security - Chapters 4-6: Cryptography** (2.5 hours) • Chapter 4: Symmetric encryption (AES) • Chapter 5: Asymmetric encryption (RSA), digital signatures • Chapter 6: TLS implementation • Pages: ~70-140
3. **OWASP ASVS v5.0 - Architecture Requirements** (1.5 hours) • URL: https://github.com/OWASP/ASVS • Focus: V1 Architecture, V10 Malicious Code, V14 Configuration

Practice (5 hours): Design enterprise security framework architecture: • Plugin system using Factory pattern for tool integration • Strategy pattern for different scanning modes (quick/full/CI) • Observer pattern for real-time alerting • Create UML diagrams and API specifications • Define data models for vulnerability tracking

Labs (4 hours): PortSwigger Insecure Deserialization (6 labs) + Prototype Pollution (4 labs, total: 95/211)

Deliverable: Framework architecture document, design patterns reference, 95 labs complete

---

Week 16: Vulnerability Aggregator Implementation (22 hours)

Python Implementation (4 hours): Build core framework implementing designed architecture

Effective Python Items 82-86 (2 hours - OPTIONAL): Community modules, virtual environments, docstrings for all code, packages for stable APIs, module-scoped configuration

Security Study (6 hours) - EXPLICIT READINGS:

1. **Secure by Design - Chapters 9-10: Failures & Cloud** (3 hours) • Chapter 9: Exception handling, designing for failures, circuit breakers, bulkheads • Chapter 10: 12-factor app security, environment configuration, logging, Three R's (Rotate, Repave, Repair)
2. **Snyk Documentation - API Integration** (1.5 hours) • URL: https://docs.snyk.io/ • Focus: REST API usage, webhook integration
3. **GitLeaks & TruffleHog Integration** (1.5 hours) • URL: https://gitleaks.io/ • URL: https://docs.trufflesecurity.com/ • Focus: CI/CD integration patterns, output formats

Practice (6 hours): Build vulnerability aggregator with multi-tool integration: • Unified interface for Semgrep, Bandit, Safety, Gitleaks • Deduplication logic for cross-tool findings • Severity normalization (CVSS mapping) • Trend analysis with historical data • REST API for dashboard integration • SQLite backend for persistence

Labs (4 hours): PortSwigger HTTP Request Smuggling (8 labs, total: 103/211)

Deliverable: Vulnerability aggregator v1.0, REST API documentation, 103 labs complete

---

Week 17: Dockerization & Documentation (22 hours)

Python Packaging (3 hours): Create proper Python package structure, setup.py/pyproject.toml, entry points

Effective Python Items 87-90 (2 hours - OPTIONAL): Root exception for API insulation, break circular dependencies, warnings for refactoring, static analysis via typing

Security Study (6 hours) - EXPLICIT READINGS:

1. **Docker Security Best Practices** (2 hours) • URL: https://docs.docker.com/develop/security-best-practices/ • Focus: Multi-stage builds, non-root users, minimal base images

2. **Trivy Container Scanning Documentation** (2 hours) • URL: https://aquasecurity.github.io/trivy/ • Focus: Image scanning, Kubernetes integration
3. **Full Stack Python Security - Chapter 12: OS Security** (2 hours) • Focus: File system security, environment variables, secrets management • Pages: ~250-275

Practice (7 hours): Dockerize security framework: • Multi-stage Dockerfile with security hardening • Docker Compose for full stack (app + database + monitoring) • Health checks and graceful shutdown • Secrets management with Docker secrets • Trivy scanning in CI/CD • Write comprehensive documentation: README, API docs, contributing guide

Writing (2 hours): Blog post: “From Script to Production: Dockerizing a Python Security Tool”

P2P Project (2 hours): Create Docker security exercise: spot the vulnerabilities in Dockerfiles

Deliverable: Dockerized security framework, complete documentation, PyPI-ready package

---

Phase 7: Portfolio & System Design (Week 18, 22 hours)

Week 18: Portfolio Polish & AppSec System Design (22 hours)

Portfolio Website Development (8 hours): • Create GitHub Pages portfolio site showcasing all projects • Security tool demos with screenshots and GIFs • Lab completion statistics and learning journey • Blog post collection with security insights • Professional bio emphasizing AppSec focus

P2P Project Public Launch Prep (4 hours): • Finalize all P2P exercises with comprehensive test suites • Create onboarding documentation for new users • Setup GitHub organization for P2P platform • Prepare marketing materials for OWASP LA presentation

AppSec System Design Practice (6 hours):

Scenario 1: Design Secure Authentication for Microservices (2 hours) • Problem: 50+ microservices need unified auth with SSO • Solution components: Identity provider, JWT tokens, API gateway validation • Security considerations: Token rotation, scope-based access, audit logging • Trade-offs: Centralized vs. distributed token validation • Document: Architecture diagram, sequence diagrams, threat model

Scenario 2: Design API Security for Multi-tenant SaaS (2 hours) • Problem: SaaS platform with 1000+ tenants, strict data isolation • Solution components: Tenant context propagation, row-level security, API rate limiting • Security considerations: BOLA prevention, tenant boundary enforcement • Trade-offs: Performance vs. isolation guarantees • Document: Data model, authorization flow, monitoring strategy

Scenario 3: Design Secrets Management for CI/CD Pipelines (2 hours) • Problem: 200+ pipelines need secure secrets without hardcoding • Solution components: HashiCorp Vault integration, dynamic secrets, rotation • Security considerations: Pipeline authentication, secret TTL, audit trails • Trade-offs: Vault availability vs. pipeline reliability • Document: Integration patterns, failure modes, disaster recovery

Intel TMT Experience Documentation: • Translate Intel work into system design language • Highlight scalability and security achievements • Prepare STAR stories for behavioral questions

Interview Prep Materials (4 hours): • Create 1-page Security Design Portfolio summarizing all three scenarios • Compile AppSec interview question bank with answers • Practice explaining complex

security concepts simply • Review OWASP ASVS for common interview topics

Security Study (0 hours - focus on practice): Reference materials: • OWASP ASVS: https://owasp.org/www-project-application-security-verification-standard/ • System Design Interview resources

Deliverable: Portfolio website live, system design portfolio, interview prep complete, P2P launch ready

---

Phase 8: Cloud Security Deep Dive (Weeks 19-20, 44 hours)

Week 19: AWS Security & IaC Scanning (22 hours)

Cloud Security Fundamentals (2 hours): Review shared responsibility model, cloud-native security principles

Security Study (10 hours) - EXPLICIT READINGS:

1. **AWS IAM Best Practices Documentation** (2.5 hours) • URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html • Focus: Least privilege, MFA enforcement, IAM Access Analyzer • Create: IAM policy templates for common use cases
2. **AWS S3 Security Best Practices** (2 hours) • URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/security-best-practices.html • Focus: Block Public Access, bucket policies, encryption (SSE-S3, SSE-KMS)
3. **AWS Lambda Security Guide** (2 hours) • URL: https://docs.aws.amazon.com/lambda/latest/dg/lambda-security.html • Focus: Function URL authentication, VPC integration, secrets management
4. **AWS API Gateway Security** (2 hours) • URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/security-best-practices.html • Focus: Lambda authorizers, Cognito integration, WAF rules
5. **Checkov IaC Scanning Documentation** (1.5 hours) • URL: https://www.checkov.io/ • Focus: Terraform rules, CloudFormation scanning, custom policies

Practice (6 hours): Build AWS security scanner: • IAM policy analyzer using boto3 (overly permissive detection) • S3 bucket configuration audit (public access, encryption) • Lambda function security review automation • Terraform/CloudFormation scanning with Checkov integration • Generate compliance report (CIS AWS Benchmark mapping)

Labs (4 hours): • AWS Free Tier security configuration exercises • Terraform security lab: fix vulnerable configurations • Practice with LocalStack for AWS API simulation

Deliverable: AWS security scanner, IaC security config templates, cloud security fundamentals mastery

Week 20: Kubernetes Security & CloudGoat (22 hours)

Container Security Fundamentals (2 hours): Review container isolation, namespace security, cgroup limits

Security Study (10 hours) - EXPLICIT READINGS:

1. **Kubernetes RBAC Documentation** (2.5 hours) • URL: https://kubernetes.io/docs/reference/access-authn-authz/rbac/ • Focus: Roles, ClusterRoles, service account security • Create: RBAC templates for common workloads
2. **Kubernetes Network Policies** (2 hours) • URL: https://kubernetes.io/docs/concepts/services-networking/network-policies/ • Focus: Ingress/egress rules, namespace isolation, CNI requirements
3. **Pod Security Standards** (2 hours) • URL: https://kubernetes.io/docs/concepts/security/pod-security-standards/ • Focus: Privileged, Baseline, Restricted profiles • Practice: Apply Pod Security Admission
4. **CloudGoat by Rhino Security Labs** (2 hours) • URL: https://github.com/RhinoSecurityLabs/cloudgoat • Focus: Setup, available scenarios, attack methodologies
5. **Trivy Kubernetes Scanning** (1.5 hours) • URL: https://aquasecurity.github.io/trivy/ • Focus: Kubernetes misconfiguration scanning, CIS benchmarks

Practice (6 hours): CloudGoat AWS exploitation labs: • iam_privesc_by_rollback: Enumerate IAM policy versions • ec2_ssrf: Exploit SSRF for metadata credentials • cloud_breach_s3: S3 data exfiltration (Capital One scenario) • codebuild_secrets: Discover plaintext secrets • Document attack paths and remediation steps

Labs (4 hours): Kubernetes security exercises: • Deploy vulnerable pods and fix with Pod Security Standards • Implement network policies for microservice isolation • RBAC hardening for multi-tenant clusters • Falco rule creation for runtime detection

Deliverable: CloudGoat completion report, K8s security templates, 103 PortSwigger labs maintained

Phase 9: DevSecOps & Modern APIs (Weeks 21-22, 44 hours)

Week 21: CI/CD Security Gates & HashiCorp Vault (22 hours)

DevSecOps Foundations (2 hours): Review shift-left security, security champions model

Security Study (10 hours) - EXPLICIT READINGS:

1. **GitHub Actions Advanced Security Features** (2.5 hours) • URL: https://docs.github.com/en/code-security/code-scanning • Focus: CodeQL custom queries, secret scanning, dependency review
2. **GitLab Security Scanning** (2 hours) • URL: https://docs.gitlab.com/user/application_security/ • Focus: SAST, DAST, container scanning, license compliance
3. **HashiCorp Vault Getting Started** (3 hours) • URL: https://developer.hashicorp.com/vault/tutorials/get-started • Focus: Secret engines, authentication methods, policies • Practice: Local Vault development setup
4. **HashiCorp Vault CI/CD Integration** (2.5 hours) • URL: https://developer.hashicorp.com/vault/docs • Focus: AppRole auth, dynamic secrets, transit encryption • Create: CI/CD Vault integration patterns

Practice (6 hours): Build advanced CI/CD security pipeline: • GitHub Actions workflow with security gates (must pass SAST, secrets scan) • Vault integration for runtime secrets injection • Dynamic database credentials with Vault • Policy-as-code with OPA/Conftest • Automated security reporting to Slack/Teams

Labs (4 hours): • Vault tutorials: secrets engines, PKI, transit encryption • GitHub Advanced Security trial exercises • Create security gate that blocks PRs with critical findings

Writing (2 hours): Blog post: “Implementing Zero-Trust CI/CD: From Hardcoded Secrets to HashiCorp Vault”

Deliverable: Production-ready CI/CD security pipeline, Vault integration patterns, blog post

---

Week 22: GraphQL Security & Service Mesh (22 hours)

Modern API Fundamentals (2 hours): Review GraphQL vs REST security differences, service mesh concepts

Security Study (10 hours) - EXPLICIT READINGS:

1. **PortSwigger GraphQL API Vulnerabilities** (2.5 hours) • URL: https://portswigger.net/web-security/graphql • Focus: Introspection attacks, batching attacks, injection through variables
2. **OWASP GraphQL Cheat Sheet** (2 hours) • URL: https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html • Focus: Query depth limiting, complexity scoring, authorization at resolver level
3. **Hacking APIs - Chapter 14: Attacking GraphQL** (2 hours) • Focus: GraphQL reconnaissance, InQL Burp extension, command injection through mutations • Pages: ~310-340
4. **Istio Security Documentation** (2 hours) • URL: https://istio.io/latest/docs/concepts/security/ • Focus: mTLS, authorization policies, peer authentication
5. **OWASP API Security - Modern Patterns** (1.5 hours) • URL: https://owasp.org/API-Security/ • Focus: API6:2023 Unrestricted Access to Sensitive Business Flows

Practice (6 hours): Build GraphQL security testing suite: • Introspection query analyzer (schema extraction) • Query complexity calculator • Batching attack detector • Injection payload fuzzer for GraphQL • Authorization bypass tester for resolvers • Integrate with existing API scanner

Labs (4 hours): PortSwigger GraphQL labs (5 labs) + Web Cache Poisoning (5 labs, total: 113/211)

Deliverable: GraphQL security scanner, service mesh security checklist, 113 labs complete

---

Phase 10: Final Preparation (Weeks 23-26, 88 hours)

Week 23: Async Python & Final Interview Prep (22 hours)

Async Python Deep Dive (4 hours): • asyncio fundamentals for concurrent security scanning • aiohttp for async HTTP requests • Build async vulnerability scanner with concurrent target testing

Security Study (6 hours) - EXPLICIT READINGS:

1. **Distributed Systems Security Patterns** (2 hours) • Focus: CAP theorem security implications, consensus security • Research: Byzantine fault tolerance basics
2. **Modern Authentication Standards** (2 hours) • WebAuthn/FIDO2 specifications overview • Passkey implementation security considerations
3. **AI/ML Security Considerations** (2 hours) • Prompt injection basics • LLM API security (rate limiting, input validation) • URL: https://portswigger.net/web-security/llm-attacks

Practice (4 hours): Build async security scanner: • Concurrent port scanning with rate limiting • Async API endpoint fuzzing • Parallel vulnerability verification • Results aggregation with asyncio.gather

Interview Preparation (6 hours): • Technical interview practice: explain OWASP Top 10, API security • System design: practice three scenarios from Week 18 • Behavioral: prepare STAR stories for security incidents • Code review: practice identifying vulnerabilities in code samples

Portfolio Finalization (2 hours): • Final updates to portfolio website • Update LinkedIn with project highlights • Prepare resume for target companies

Deliverable: Async scanner, interview prep complete, portfolio finalized

---

Week 24: Tool Polish & Documentation (22 hours)

Complete Unfinished Tools (10 hours): • Review all tools built during curriculum • Fix bugs, add error handling, improve UX • Ensure consistent coding style across all projects • Add type hints and run mypy • Achieve 80%+ test coverage on core modules

Documentation Enhancement (8 hours): • Comprehensive README files for each project • API documentation with examples • Architecture decision records (ADRs) for major design choices • Video demos for complex tools • Contributing guidelines for open-source projects

GitHub Profile Optimization (4 hours): • Pin best 6 repositories • Create organization for security tools • Setup GitHub Actions for all repositories • Add badges (build status, coverage, license) • Write compelling repository descriptions

Deliverable: All tools production-ready, documentation complete, GitHub profile optimized

---

Week 25: Advanced Interview Prep & Mock Interviews (22 hours)

Advanced Interview Preparation (8 hours): • Deep dive into target companies' security challenges • Research recent security incidents and how you'd prevent them • Prepare questions to ask interviewers about security culture • Review AppSec engineer job descriptions for skill alignment

Mock Interviews (8 hours): • Schedule 3-4 mock interviews with peers or mentors • Practice whiteboard security architecture • Code review exercises with timed vulnerability identification • System design interviews with security focus

System Design Practice (6 hours): Additional scenarios: • Design secure file upload system for enterprise • Design fraud detection pipeline with security controls • Design secure API gateway for partner integrations • Practice explaining trade-offs verbally

Deliverable: Mock interview feedback incorporated, confidence in all interview formats

---

Week 26: Active Job Applications & Networking (22 hours)

Active Job Applications (10 hours): • Apply to 2-3 positions daily at target companies • Customize cover letter for each application • Track applications in spreadsheet with follow-up dates • Priority targets: Trail of Bits, NCC Group, Anthropic, GitLab, Stripe, Coinbase

Networking (8 hours): • Attend OWASP LA chapter meeting • Visit Null Space Labs hackerspace • Connect with AppSec professionals on LinkedIn • Reach out to security engineers at target companies • Share blog posts in security communities (Twitter/X, InfoSec community)

Final Portfolio Polish (4 hours): • Last updates based on any feedback • Ensure all links work • Test all demos • Prepare 30-second and 2-minute elevator pitches

Deliverable: 15+ applications submitted, network expanded, ready for interviews

---

Curriculum Summary

Total Hours by Phase

Phase	Weeks	Hours
Phase 1: Foundation	1-3	66
Phase 2: Exploitation	4-6	66
Phase 3: Authentication & Authorization	7-9	66
Phase 4: Security Automation	10-12	66
Phase 5: Research & Discovery	13-14	44
Phase 6: Enterprise Production Tools	15-17	66
Phase 7: Portfolio & System Design	18	22
Phase 8: Cloud Security Deep Dive	19-20	44
Phase 9: DevSecOps & Modern APIs	21-22	44
Phase 10: Final Preparation	23-26	88
Total	**26 weeks**	**572 hours**

Lab Progress Target

Milestone	Week	Labs Complete
Foundation Complete	3	26/211
Exploitation Complete	6	55/211
Auth & Authz Complete	9	75/211
Automation Complete	12	95/211
Research Complete	14	85/211
Enterprise Complete	17	103/211
Cloud Complete	20	103/211
DevSecOps Complete	22	113/211
Curriculum Complete	26	120+/211

Key Deliverables

Security Tools Portfolio: 1. SQL Injection Detection Script 2. XSS Payload Generator with Encoding Decorators 3. Credential Analysis Tool 4. IDOR/Authorization Scanner 5. CSRF PoC Generator 6. OAuth Flow Analyzer Package 7. Secure Authentication Module 8. RBAC/ABAC Framework 9. SAST Orchestrator 10. CI/CD Security Scanner 11. Security Framework Dashboard 12. API Security Scanner 13. Vulnerability Aggregator 14. AWS Security Scanner 15. GraphQL Security Testing Suite 16. Async Concurrent Scanner

Blog Posts: 1. “Building Your First Security Testing Toolkit in Python: Lessons from 55 Labs” 2. “5 Authorization Bugs AI Code Generation Consistently Misses” 3. “Building a Python Security Dashboard: Integrating 5 Tools in 200 Lines” 4. “What I Learned Auditing PyJWT: A Junior Security Engineer’s Perspective” 5. “From Script to Production: Dockerizing a Python Security Tool” 6. “Implementing Zero-Trust CI/CD: From Hardcoded Secrets to HashiCorp Vault”

P2P Exercises: 15+ exercises with 500+ automated tests

Certifications to Pursue: • Burp Suite Certified Practitioner (BSCP) - after completing 180+ labs • AWS Security Specialty (optional, if cloud focus)

Resource Quick Reference

Primary Books: • Python Workout, Second Edition - Reuven Lerner (Manning) • Effective Python, 2nd Edition - Brett Slatkin (Addison-Wesley) • API Security in Action - Neil Madden (Manning) • Hacking APIs - Corey Ball (No Starch Press) • Full Stack Python Security - Dennis Byrne (Manning) • Secure by Design - Johnsson, Deogun, Sawano (Manning)

Primary Online Resources: • PortSwigger Web Security Academy: https://portswigger.net/web-security • OWASP Top 10: https://owasp.org/Top10/ • OWASP API Security Top 10: https://owasp.org/API-Security/ • OWASP Cheat Sheet Series: https://cheatsheetseries.owasp.org/ • OWASP ASVS: https://owasp.org/www-project-application-security-verification-standard/ • OWASP Juice Shop: https://owasp.org/www-project-juice-shop/ • Semgrep: https://semgrep.dev/docs/ • HashiCorp Vault: https://developer.hashicorp.com/vault/tutorials

Practice Environments: • PortSwigger Labs (211+ labs) • OWASP Juice Shop (100+ challenges) • CloudGoat (10+ AWS scenarios) • OWASP WebGoat (educational platform)