

Command & Code Reference

Network Forensics Framework — Step-by-Step Commands

*Test-bed: Kali Linux (192.168.56.102) attacker • Metasploitable2 (192.168.56.101) target • VirtualBox
host-only vboxnet0*

This document collects every command, display filter and script used across the practical demonstration (Chapter 4 and the Appendices) in one place, in the order they are run. All work is performed inside an isolated host-only lab, with no path to any external network.

Phase 1 — Environment Setup

The lab is two VirtualBox guests on a single host-only adapter (vboxnet0), so every packet stays confined to the two VMs with no route to the host network or internet.

Step 1: Configure the isolated network (in VirtualBox GUI)

- For the Kali guest, open Settings → Network → Adapter 1 and set:
- Attached to: Host-Only Adapter
- Name: vboxnet0
- Promiscuous Mode: Deny (VirtualBox default)
- Apply the same host-only adapter to the Metasploitable2 guest.

Step 2: Verify the host-only network exists (optional, host terminal)

```
# List VirtualBox host-only networks
$ VBoxManage list hostonlyifs

# Confirm the vboxnet0 subnet / mask
$ ip addr show vboxnet0
```

Phase 2 — Traffic Generation & Reconnaissance

Before touching the target, the investigating host verifies its own identity and tool-chain (a deliberate negative control), then reconnaissance is run against the target.

Step 3: Confirm the investigating host address

```
$ hostname -I  
192.168.56.102
```

Step 4: Baseline reachability check over vboxnet0

```
$ ping -c 4 192.168.56.102  
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data.  
4 packets transmitted, 4 received, 0% packet loss, time 3219ms  
rtt min/avg/max/mdev = 0.034/0.041/0.047/0.005 ms
```

Step 5: Self-scan smoke test (validate the Nmap install against a known host)

```
# Default TCP scan of the investigator's own host  
$ nmap 192.168.56.102  
Nmap scan report for 192.168.56.102  
All 1000 scanned ports on 192.168.56.102 are in ignored states.  
Not shown: 1000 filtered tcp ports (no-response)  
  
# Service / version-detection self-scan  
$ nmap -sV 192.168.56.102  
Nmap done: 1 IP address (1 host up) scanned in 202.65 seconds
```

Note: Running this against the investigator's own address first makes Nmap's timing and "filtered/closed/open" behaviour easy to interpret before moving to a target under investigation.

Step 6: Reconnaissance against the target (Metasploitable2)

```
# TCP SYN (stealth) scan of the target  
$ nmap -sS 192.168.56.101  
  
# Service / version detection on the target  
$ nmap -sV 192.168.56.101
```

Note: The SYN scan produces the port-scan signature analysed in Section 4.2.2 — many destination ports from one source within well under a second.

Phase 3 — Packet Capture

Wireshark 4.6.6 captures on the eth0 interface (the adapter bound to vboxnet0) while the reconnaissance above is under way.

Step 7: Start the live capture (Wireshark GUI)

- Open Wireshark → select the eth0 interface → Start capture (the blue shark-fin / double-click).

Step 8: Equivalent command-line capture (optional)

```
# Capture on eth0 and write a pcapng file (command-line equivalent)
$ sudo dumpcap -i eth0 -w network_forensic_capture.pcapng

# Or with tshark
$ sudo tshark -i eth0 -w network_forensic_capture.pcapng
```

Note: Stop the capture (Ctrl+C, or the red square in Wireshark) once the reconnaissance run is complete.

Phase 4 — Packet Analysis (Wireshark Display Filters)

These display filters were typed into the Wireshark filter bar to produce the Section 4.2 results. Each filter is applied to the single saved capture.

Display Filter	Purpose
<code>tcp.flags.syn==1</code>	Isolate SYN and SYN/ACK packets to reconstruct the scan (Fig. 4.13)
<code>udp.stream eq 2</code>	Follow one specific UDP conversation — WS-Discovery (Fig. 4.10)
<code>tcp.stream eq 1005</code>	Follow one specific TCP conversation — HTTP stream (Figs. 4.14–4.15)
<code>dns</code>	Isolate DNS name-resolution traffic — version.bind probe (Fig. 4.12)
<code>ip.addr==192.168.56.101</code>	Isolate all traffic to/from the target host

Step 9: Right-click any packet → Follow → TCP/UDP/HTTP Stream to reassemble a full conversation

- Used to reconstruct the HTTP GET / exchange on port 80 and the WS-Discovery SOAP probe.

Step 10: Equivalent command-line analysis (optional)

```
# Pull all SYN/SYN-ACK packets from the saved file
$ tshark -r network_forensic_capture.pcapng -Y "tcp.flags.syn==1"

# Follow a specific TCP stream as text
$ tshark -r network_forensic_capture.pcapng -q -z
follow,tcp,ascii,1005
```

Phase 6 — Evidence Collection & Integrity Verification

Once the capture is complete it is moved into a dedicated case folder, ownership is restricted to the investigating account, and a SHA-256 digest is computed and logged — establishing a verifiable chain of custody.

Step 11: Create the case-evidence directory and fix ownership

```
# Inspect any pre-existing (root-owned) evidence directory
$ ls -ld /home/kali/forensic-evidence

# Create the case folder under the investigator's home
$ mkdir -p /home/kali/forensic-evidence

# Restrict ownership to the investigating account
$ sudo chown -R kali:kali /home/kali/forensic-evidence

# Confirm size and permissions of the relocated capture
$ ls -lh /home/kali/forensic-evidence
```

Step 12: Compute and log the SHA-256 digest (chain of custody)

```
# Fingerprint the capture and save the digest alongside it
$ sha256sum network_forensic_capture.pcapng | tee evidence-hash.txt
```

Note: Recompute this digest and compare it against evidence-hash.txt before every future analysis session to prove the file has not been altered.

Appendix — Traffic-Generation Orchestration Script

Section 3.3.3 describes a shell/Python orchestration script that drives each experimental run under identical, repeatable conditions: it (i) generates baseline normal traffic, (ii) launches one attack scenario at a scheduled point, (iii) writes a timestamped CSV log entry at the start and end of the attack, and (iv) runs identically with NTP-synchronised clocks for every repetition. A reference implementation matching that description is below.

```
#!/usr/bin/env bash
# run_scenario.sh — orchestrates one repeatable experimental run
# Usage: sudo ./run_scenario.sh <scenario_name>

TARGET="192.168.56.101"
LOG="attack_log.csv"
SCENARIO="${1:-port_scan}"

# Create the CSV header once
[ -f "$LOG" ] || echo "scenario,event,timestamp_iso" > "$LOG"

log() { echo "$SCENARIO,$1,$(date --iso-8601=seconds)" >> "$LOG"; }

# (i) Baseline normal traffic for realistic background noise
ping -c 4 "$TARGET" > /dev/null 2>&1
curl -s "http://$TARGET/" > /dev/null 2>&1

# (ii) + (iii) Launch the chosen scenario, logging start and end
log "attack_start"
case "$SCENARIO" in
    port_scan) nmap -sS -sV "$TARGET" ;;
    *)          echo "Unknown scenario: $SCENARIO" ;;
esac
log "attack_end"

echo "Run complete. See $LOG for timestamps."
```

Note: Only the Port Scan scenario was executed in the interim demonstration (Chapter 4). The other scenarios named in Table 3 are scoped for the full repeat-measures campaign (Section 3.3.7).

Quick Command Index

Command	What it does
hostname -I	Confirm investigating host address
ping -c 4 192.168.56.102	Baseline reachability / RTT
nmap 192.168.56.102	Default TCP self-scan (smoke test)
nmap -sV 192.168.56.102	Version-detection self-scan
nmap -sS 192.168.56.101	SYN stealth scan of target
sudo dumpcap -i eth0 -w <file>	Live packet capture
tcp.flags.syn==1	Wireshark filter: isolate scan SYNs
dns	Wireshark filter: DNS traffic
mkdir -p /home/kali/forensic-evidence	Create case folder
sudo chown -R kali:kali <dir>	Restrict evidence ownership
sha256sum <file> tee evidence-hash.txt	Compute & log integrity digest