Undergraduate Project Report

“Penetration Testing of Android Devices”

Supervised by: - Prof. Gaurav Tiwari, Prof. Subhajit Roy
Submitted by:- Divyanshu Sharma(220381)

1.) Network Traffic Sniffing using Bettercap

1. Introduction

This undergraduate project utilizes Bettercap, a versatile and powerful network monitoring and attacking tool, to demonstrate penetration testing techniques in a controlled environment. The project focuses on network traffic sniffing and ARP spoofing to analyze vulnerabilities in a wireless network.

2. Tools Used

- **Bettercap**: A comprehensive network attack and monitoring tool.
- **Python**: For automating interactions with Bettercap using a script.
- **Configuration File**: A JSON file for customizable actions and settings.

3. Bettercap Overview

Bettercap is utilized to perform network sniffing and ARP spoofing. It operates over the **wlan0** interface to demonstrate real-time network traffic monitoring and analysis.

4. Configuration and Usage

Configuration File

The config_bettercap.json file defines the interactions between the Python script and Bettercap. It contains the following key elements:

- **Program**: sudo bettercap -iface wlan0
- **Start Interactions**:
  - **net.probe on**: Initiates probing of the network.
  - **net.show**: Displays active network devices.
  - **set arp.spoof.targets <IP>**: Sets the target IP for ARP spoofing.
  - **arp.ban on**: Blocks ARP responses for better traffic interception.
  - **arp.spoof on**: Initiates ARP spoofing to intercept traffic.
  - **net.sniff on**: Starts network sniffing to capture live data.
- **Stop Interactions**:
  - **arp.spoof off**: Stops ARP spoofing.
  - **net.sniff off**: Terminates network sniffing.

Automation Script

The Python script “driver_bettercap2.py” interacts with Bettercap using the pexpect library. Key features of the script include:

1. **Execution of Configured Commands**: Reads and executes commands defined in config_bettercap.json.
2. **ARP Spoof Target Customization**: Dynamically replaces the <IP> placeholder with the target IP.
3. **Logging**: Captures and logs real-time sniffing output to a file named sniffing_output.log.

5. Methodology

1. **Setup**:
   a. Install Bettercap on a system with a wireless interface (wlan0).
   b. Ensure the target machine is on the same network.

2. **Command Execution**:
    a. Run the automation script: **python3 driver_bettercap2.py <Target IP>**
    b. The script sets up the environment and executes the interactions defined in the configuration file.
3. **Network Sniffing**:
    a. Captures live traffic data and saves it to a log file for analysis.
4. **Stopping Interactions**:
    a. Ensures proper cleanup by disabling ARP spoofing and stopping sniffing.

6. Results

- Real-time network traffic was successfully intercepted and logged using Bettercap.
- The automation script facilitated seamless execution and ensured modularity for testing different configurations.

7. Conclusion

The integration of Bettercap with a Python automation script demonstrates a practical approach to penetration testing. It highlights potential vulnerabilities in wireless networks and provides a foundation for further exploration of network security measures.

2.) DNS Spoofing Using Ettercap

1. Introduction

This undergraduate project demonstrates penetration testing techniques using Ettercap, a robust tool for network monitoring and attacks. The focus of the project is on DNS spoofing and ARP spoofing in a controlled network environment to analyze vulnerabilities in DNS infrastructure.

2. Tools Used

- **Ettercap**: A comprehensive tool for network attacks, including DNS spoofing and ARP poisoning.
- **Python**: For automating interactions with Ettercap using a script.
- **Configuration File**: A JSON file defining network settings and Ettercap options.

3. Ettercap Overview

Ettercap was used to perform DNS spoofing and ARP spoofing, leveraging the **wlan0** interface for real-time network traffic interception. Traffic for the target domain **www.amazon.com** was redirected to a spoofed server, demonstrating the potential impact of DNS attacks.

4. Configuration and Usage

Configuration File

The config_dns.json file outlines the network interface and Ettercap log file path. Key elements include:

- **interface**: Specifies the network interface (e.g., wlan0).
- **log_file**: Defines the Ettercap log file for capturing events.

Commands in Python Script

The Python script driver_dns.py automates interactions with Ettercap. Key functionalities include:

1. **Updating DNS Configurations**: Updates the **etter_dns** file with the spoofed IP for the target domain.
2. **Apache Server Integration**: Ensures the spoofed domain resolves to the attacker's machine.
3. **Automated Ettercap Execution**: Executes DNS spoofing and monitors real-time logs for success.

5. Methodology

Setup:

1. Install Ettercap on a system with a wireless interface (e.g., wlan0).
2. Configure Apache server to host the spoofed website.
3. Update config_dns.json with the correct interface and log file paths.

Command Execution:

1. Run the script: **python3 driver_dns.py <Target IP 1> <Target IP 2>**
   This initializes Ettercap, starts ARP spoofing, and monitors for DNS spoofing success.

2. Monitor Ettercap logs in real time to detect access to the spoofed domain.

Network Spoofing:

- Redirects traffic for the domain **www.amazon.com** to the attacker's machine, demonstrating DNS spoofing.

Stopping Interactions:

- Ettercap is terminated after confirming spoofing success, and logs are saved for analysis.

6. Results

DNS spoofing was successfully demonstrated using Ettercap. The logs indicate that traffic intended for **www.amazon.com** was redirected to the attacker-controlled machine. The Python script facilitated automation, ensuring seamless configuration and execution.

7. Conclusion

The integration of Ettercap with Python automation highlights practical methods for executing DNS spoofing attacks. This project underscores the vulnerabilities in DNS infrastructure and ARP protocols while providing insights for designing robust countermeasures against such attacks.

(3.) Cracking Wi-Fi Password using Aircrack-ng

1. Introduction

This project demonstrates the use of **Aircrack-ng**, a powerful suite of tools for wireless network penetration testing. The primary focus is on capturing network handshakes and cracking Wi-Fi passwords. The process is automated using Python scripts to simplify configuration and execution in a controlled environment.

2. Tools Used

- **Aircrack-ng**: For wireless network monitoring, packet capture, and password cracking.
- **Python**: To automate Aircrack-ng operations and streamline the workflow.
- **Airmon-ng**: A utility within the Aircrack-ng suite for enabling monitor mode on wireless interfaces.
- **Airodump-ng**: A tool for capturing network packets and identifying target networks.
- **RockYou.txt**: A commonly used wordlist for password cracking.

3. Aircrack-ng Overview

Aircrack-ng was used for the following:

1. Enabling monitor mode on a wireless network interface (wlan0).
2. Scanning for networks to locate the target SSID.
3. Capturing WPA handshake packets for the target SSID.
4. Cracking the Wi-Fi password using a dictionary attack.

4. Configuration and Usage

Python Script

The Python script **driver_aircrack.py** automates the following:

- Enabling monitor mode on the wireless interface.
- Scanning and identifying the target network based on its SSID.
- Capturing WPA handshakes using **Airodump-ng.**
- Launching **Aircrack-ng** for password cracking with a wordlist.

Key Script Details:

- **Target SSID**: Hardcoded in the script as "OnePlus Nord CE 2 Lite 5G" which is our target network.
- **Log File**: **capture_log.txt** captures the output and logs of the script's execution.

5. Methodology

Setup:

1. Install the Aircrack-ng suite on the system.
2. Ensure the target SSID is within range.
3. Update the script as needed to match the wireless interface (default: wlan0).

Execution Steps:

1. **Monitor Mode**: The script uses **airmon-ng** to enable monitor mode on wlan0.
2. **Network Scan**: **Airodump-ng** scans for available networks, and the script extracts the BSSID and channel of the target SSID.
3. **Handshake Capture**: **Airodump-ng** captures WPA handshakes for the target network.
4. **Password Cracking**: **Aircrack-ng** attempts to crack the captured handshake using the **RockYou.txt** wordlist.

6. Results

- The script successfully located the target SSID and captured **WPA handshakes.**
- The password cracking step was partially **automated**, with real-time feedback provided in the terminal and logs.
- Results were saved for future analysis in both .csv and .cap files.

7. Conclusion

This project highlights the practical use of Aircrack-ng for Wi-Fi penetration testing and the power of Python for automating repetitive tasks. The methodology underscores the vulnerabilities of WPA-protected networks when weak passwords are used, emphasizing the need for robust security practices in wireless networking.