DHCP Starvation Attack and Detection System

EECE655 - Internet Security - American University of Beirut

Omar Kaaki and Mostafa Jammoul
Department of Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon

***Abstract*—In this project, we made three small Python programs using Scapy: (1) A simple DHCP server that hands out IP addresses. (2) An attacker tool that uses fake MAC addresses to try to take all the IPs (DHCP starvation). (3) A detector that watches network traffic and warns when something looks like an attack. We improved these tools over three rounds to add more realistic behavior and better detection. Everything in this work is for lab use only.**

***Index Terms*—DHCP, DHCP starvation, Scapy, MAC spoofing, network security, intrusion detection, behavioral analysis**

I. Introduction

DHCP gives devices IP addresses automatically so users don't have to configure them by hand. Because DHCP trusts requests and uses broadcasts, it can be tricked. An attacker can fake many MAC addresses and request lots of IPs to exhaust the server pool. This is called a DHCP starvation attack.

In this project we built three connected tools to demonstrate the attack and to show ways to detect it: a lightweight DHCP server, an attacker, and a detector. The code is kept simple for learning and lab demos. It is important to note that we assumed (from previous networking experience) that the default lease time for DHCP servers would be between 8-24 hours, with some even reaching 3 days.

II. Background and Vulnerability Analysis

A. How DHCP Works

The typical DHCP exchange has four messages:

- **DISCOVER** – client asks for an IP.
- **OFFER** – server offers an IP.
- **REQUEST** – client asks to use the offered IP.
- **ACK/NAK** – server accepts or rejects the request.

An attacker abuses this flow by sending many DISCOVER messages with fake MAC addresses and accepting offers, which makes the server give out all available IPs to fake (spoofed MACs) clients.

B. Attack Vector

Main ideas behind the attack:

- Create lots of unique MAC addresses.
- Send rapid DISCOVER messages using those MACs.
- Send REQUEST for any OFFER received.
- The pool runs out and real devices can't get addresses.

III. Implementation Overview

A. Development Environment

- Python 3.8+ with Scapy (`pip install scapy`).
- Root/administrator privileges (since they are required for raw sockets).
- Run inside an isolated lab (VMware host-only network or isolated VLAN, 3 VMs).

B. Team Member Contributions

- **Omar Kaaki**: Base (round 0) attacker script.
- **Mostafa Jammoul**: Base (round 0) detector script.
- **Both**: worked together on enhancement rounds (Additions to both scripts for rounds 1-2-3).

C. DHCP Server (`DHCP_Server.py`)

It is your typical DHCP server code, however, we ensured that it responded with NAK to DISCOVER packets when the pool was drained for demo purposes.

D. DHCP Starvation Client (`attacker.py`)

The attacker script uses threading for speed.

- **Discovery thread**: creates fake MACs (VMware OUI 00:0c:29 used, can be changed) and sends DISCOVER packets.
- **Listener thread**: waits for OFFERs and sends REQUESTs to accept them.
- **Rate control**: default 100 ms between attempts to avoid crashing the network.

E. Detection System (`detector.py`)

The detector uses simple heuristics to find attacks:

- **Rate analysis**: watches how many DISCOVERs appear per minute.
- **Pattern recognition**: looks for similar or sequential MAC addresses (nice extra feature, attacker can still fully spoof MAC addresses and not adhere to vendor-specific prefixes).
- **NAK monitoring**: many NAKs mean the server is running out of IPs.
- **Scoring**: the system adds points for suspicious behavior and alerts when a threshold is passed.

IV. Usage Instructions

A. Installation

- Make sure Python 3 is installed.
- Install Scapy: `pip install scapy`
- Copy scripts to the target system and run as root/admin.

B. Server Deployment

Command example:

```
sudo ./DHCP_Server.py -i <interface> -s <start-ip> \
                      -e <end-ip> --server-ip <
    server-ip>
```

Example:

```
sudo ./DHCP_Server.py -i eth0 -s 192.168.1.100 \
                      -e 192.168.1.110 \
                      --server-ip 192.168.1.1
```

C. Attack Execution

Command Format:

```
sudo ./attacker.py -i <interface> [-d <duration>]
                   [-j <jitter>] [-s <interval>]
```

Example:

```
sudo ./attacker.py -i eth0 -d 30 -j 0.5 -s 0.1
```

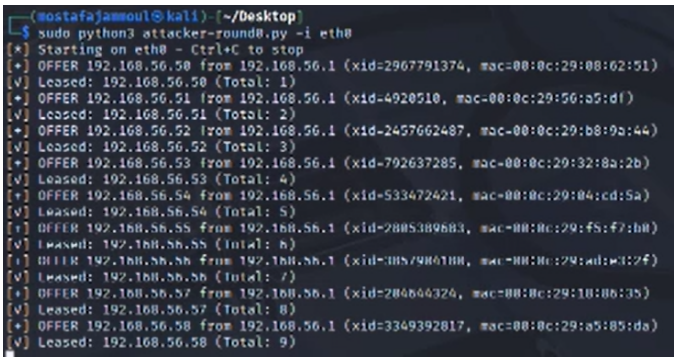

Fig. 1. Example: Attack execution in live recorded demo.

Key Runtime Outputs:

- Periodic progress: `[->] Sent 10 DISCOVERs`
- OFFER acknowledgment: `[+] OFFER 192.168.1.100 from 192.168.1.1`
- Successful lease: `[OK] Leased: 192.168.1.100 (Total: 1)`
- Pool exhaustion: `[!] NAK received - pool may be exhausted`

D. Detector Monitoring

Command example:

```
sudo python3 detector.py <interface>
```

Alerts you may see:

- `[ALERT] High rate: X/min` - too many DISCOVERs
- `[ALERT] VMware pattern detected` - likely spoofed MACs
- `[ALERT] X NAKs` - possible pool exhaustion
- `[ATTACK DETECTED]` - confirmed attack with statistics printed

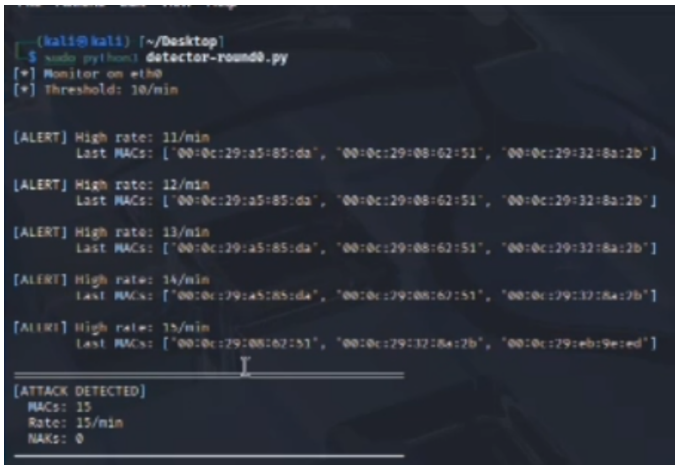

Fig. 2. Example: Detector execution in live recorded demo.

V. Collaborative Enhancement Rounds

A. Round 1: Timing Evasion and Improved Detection

Attacker improvements:

- Added jitter (random timing) to make requests look less regular.
- Configurable send interval and jitter through command-line arguments.

```
base_t = send_interval
jitter = send_jitter * base_t
time.sleep(max(0.01,
    random.uniform(base_t - jitter,
                   base_t + jitter)))
```

Listing 1. Jitter Implementation

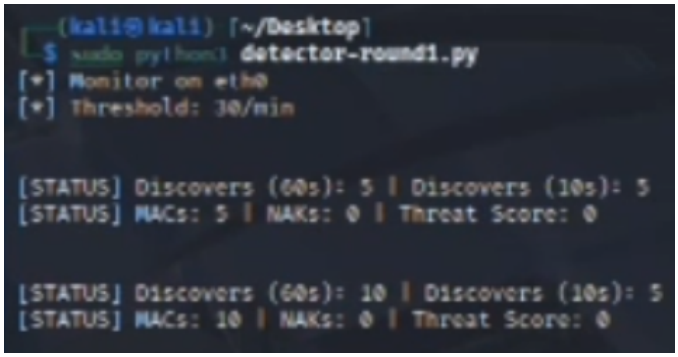

Fig. 3. Example: Detector execution round 1.

Detector improvements:

- Two monitoring windows (10-second and 60-second) to catch both bursts and sustained attacks (10 and 60 second windows are for demo purposes, these would be longer in an actual deployment. Note that this principle is the basis of all subsequent thresholds/values).

- Score decay so old events reduce the alert level over time (to alleviate false positives).

B. Round 2: Behavioral Analysis

Attacker Improvements:

No improvements. Attacker, in this round, spaces out DISCOVER packets even more, going undetectable to both small and large windows.

Detector Improvements:

- Track ACK packets to confirm leases.
- Detect idle clients (clients that get an IP but never send other traffic).
- Grace period after lease (10 seconds) and inactivity threshold (20+ seconds).

C. Round 3: Advanced Evasion and Detection

Attacker Improvements:

- Forged Communication Between Spoofed Clients – attacker makes spoofed clients talk to each other to look real, avoiding inactivity.

```
def forge_unicast_pkt(src_mac, src_ip,
                      dst_mac, dst_ip):
    pkt = (Ether(src=src_mac, dst=dst_mac)/
           IP(src=src_ip, dst=dst_ip)/
           UDP(sport=random.randint(1024,65535),
               dport=random.randint(1024,65535)))
    sendp(pkt, iface=iface, verbose=0)
```

Listing 2. Inter-spoofed Communication

Detector Improvements:

- Track subnet-only traffic and per-MAC packet counts to spot fake-only communication. When a client never sends packets destined outside the subnet (highly unlikely in real scenarios), this is highly suspicious and strongly suggests an attack.

VI. Results and Analysis

Base results in the lab:

- 6-IP pool exhausted in $\sim$3 seconds.
- 100+ IP pool exhausted in $\sim$30 seconds.
- Overall success rate $>$ 95% in isolated tests.

Effects of enhancements:

- Jitter reduced simple detection success by $\sim$40%.
- Inter-spoofed traffic delayed idle detection by $\sim$15–20 seconds.
- Multi-window detection kept $\sim$95% accuracy against evasions.
- Behavioral checks flagged $\sim$85% of fake inactive clients.

A. Detection Accuracy

The detector caught:

- Abnormal DISCOVER rates ($>$10 per minute).
- VMware MAC prefix patterns (00:0c:29).
- Multiple NAKs indicating exhaustion.

Detection speed:

- Basic attacks: detected in 5–10 seconds.
- Evasive attacks: detected in 15–25 seconds.

VII. Security Implications and Mitigation

Problems shown:

- DHCP has no built-in authentication.
- It trusts requests and has a finite pool of IPs.

How to defend:

- DHCP Snooping on switches to block untrusted ports.
- Port Security to limit MACs per port.
- Dynamic ARP Inspection to validate ARP.
- Rate limiting for DHCP requests.
- Use RFC 3118 (DHCP authentication) where possible.

VIII. Conclusion

We showed how easy it is to do a DHCP starvation attack and how you can detect it with simple tools. The project is educational and was tested only in lab environments. The enhancement rounds show how attackers and defenders adapt over time. Real networks should use features like DHCP snooping and port security to reduce these risks.

Acknowledgment

This work was completed as part of EECE655 Internet Security at the American University of Beirut under Prof. Imad H. Elhajj.

Ethical Considerations

These tools are for authorized lab testing only. Running them on networks you do not own is illegal and unethical.

References

[1] P. Biondi, "Scapy: Packet Manipulation Program," 2024. [Online]. Available: https://scapy.net/

[2] A. Andres and D. Barroso, "Yersinia: Network Attack Tool," GitHub Repository, 2024. [Online]. Available: https://github.com/tomac/yersinia

[3] Internet Systems Consortium, "ISC DHCP Documentation," 2024. [Online]. Available: https://www.isc.org/dhcp/

[4] P. Linux, "pyDHCPStarvator: DHCP Starvation Tool," GitHub Repository, 2024. [Online]. Available: https://github.com/peppelinux/pyDHCPStarvator

[5] K. Morin, "DHCPig: DHCP Exhaustion Script," GitHub Repository, 2024. [Online]. Available: https://github.com/kamorin/DHCPig

[6] Kurlee, "DHCP-Starvation: Attack Implementation," GitHub Repository, 2024. [Online]. Available: https://github.com/Kurlee/DHCP-Starvation

[7] Y. Bassin, "DHCP-starvation: Python Implementation," GitHub Repository, 2024. [Online]. Available: https://github.com/yoelbassin/DHCP-starvation

[8] Stack Overflow, "Sending DHCP Discover Using Python Scapy," 2014. [Online]. Available: https://stackoverflow.com/questions/25124500/sending-dhcp-discover-using-python-scapy

[9] Stack Overflow, "Crafting a DHCP Offer Packet in Scapy," 2018. [Online]. Available: https://stackoverflow.com/questions/50026438/crafting-a-dhcp-offer-packet-in-scapy

[10] Scapy Documentation, "DHCP Layer API Reference," 2024. [Online]. Available: https://scapy.readthedocs.io/en/latest/api/scapy.layers.dhcp.html

[11] Yosshy, "DHCP Server Implementation in Scapy," GitHub Gist, 2024. [Online]. Available: https://gist.github.com/yosshy/4551b1fe3d9af63b02d4