MICS Open Network Security

November 9, 2025

1 Introduction

This document contains practical tasks for the DNS Covert Channels lab. You will act as a detective investigating suspicious DNS traffic from a compromised agent machine. The agent is exfiltrating data through DNS queries using various covert channel techniques.

Estimated total time: 55 - 60 minutes

Prerequisites:

- Lab environment running (see Ethical Hacking Guide)
- Basic understanding of DNS protocol
- Familiarity with command-line tools (tshark, python)
- Access to both DNS-DHCP VM and Agent VM

2 Task 1: Familiarizing with the Lab Environment

Estimated time: 10 - 15 minutes

Before beginning the covert communication analysis, take a few minutes to explore the lab environment and verify that the virtual network is working correctly. This warm-up task helps you understand how the Agent and DNS-DHCP VMs interact and gives you freedom to experiment safely - if anything breaks, simply rerun `launch-lab` to reset the setup.

2.1 Connecting to the Virtual Machines

Setup – Terminal 1 (DNS-DHCP VM):

```
cd DNS
vagrant ssh
```

Setup – Terminal 2 (Agent VM):

```
cd Agent
vagrant ssh
```

Goal: Two terminals are open, one for the DNS Server and one for the Agent.

2.2 Testing Network Connectivity

From the **Agent VM**, confirm that the DNS Server is reachable:

```
ping 192.168.10.1
```

From the **DNS Server**, list its network interfaces:

```
ip addr
```

Goal: The Agent should receive replies from 192.168.10.1, confirming basic connectivity.

2.3 Verifying DNS Resolution

From the **Agent VM**, perform a few DNS lookups through the server:

```
nslookup example.com 192.168.10.1
dig example.com @192.168.10.1
```

Goal: Both commands should return a valid IP address, confirming that `dnsmasq` is active on the DNS Server.

2.4 Exploring Configuration and Services

On each VM:

```
ip addr
cat /etc/resolv.conf
```

On the **DNS Server**:

```
sudo systemctl status dnsmasq
```

Goal: Identify which interface is used for communication, confirm the configured DNS resolver, and verify that `dnsmasq` is running.

2.5 Breaking and Fixing the Network (Experiment)

On the **DNS Server**, temporarily stop the DNS service:

```
sudo systemctl stop dnsmasq
```

From the **Agent VM**, try another DNS lookup – it should now fail. Then restart the service:

```
sudo systemctl start dnsmasq
```

Goal: Observe how service availability affects network behavior and learn what happens when the server stops responding.

If network communication is still unstable, rerun:

```
launch-lab
```

to restore the environment.

2.6 Static IP Conflict Experiment

Note: If there is aprox. 45 minutes of class left, feel free to skip this step as it is time consuming.

On the **Agent VM**, try assigning the same IP address as the DNS Server to the `eth1` interface:

```
sudo ip addr flush dev eth1
sudo ip addr add 192.168.10.1/24 dev eth1
```

Now test DNS resolution again:

```
dig example.com @192.168.10.1
```

Then restart the lab setup:

```
launch-lab
```

Observation: The lookup will fail because both machines now claim the same IP address, creating a network conflict that prevents the real DNS Server from receiving queries.

2.7 Reflection Questions

Question 1.1: What IP address is assigned to the DNS Server, and how does the Agent obtain its IP address under normal conditions?

Question 1.2: Which service handles DNS requests on the server?

Question 1.3: After stopping `dnsmasq`, how did the Agent's behavior change?

Question 1.4: What happened when the Agent used the same IP address as the DNS Server?

Question 1.5: Why is it important to experiment with network failures before beginning covert communication analysis?

3 Task 2: Single-Message Subdomain Encoding

Estimated time: 20 minutes

In this task, you will investigate single-packet covert channels where data is encoded directly in DNS subdomain labels. The agent encodes entire messages into a single DNS query subdomain.

Important Note - Live vs. Saved Captures:

- Most exercises use **live tshark sessions** for immediate feedback and simpler workflow
- We filter captures to show **only DNS request packets** (dns.flags.response == 0) for cleaner data analysis
- This filtering removes DNS response packets, making it easier to focus on the queries being sent
- You can also **save captures to pcap files** for later analysis or submission (demonstrated in Task 1.4c)
- Pcap files can be transferred from the VM to your host machine for archiving or grading

Important Note - Using the DNS Covert Channel Tools: This lab provides ready-to-use commands for running the DNS covert channel agent and decoder tool. You will not need to write code from scratch. However, Task 1.6 includes a small coding exercise to help you understand the encoding mechanisms better. The codebase includes a README file with detailed usage instructions for both the agent and decoder tool if you need to reference them during the lab.

3.1 Task 1.1: Understanding Normal DNS Traffic

Before investigating covert channels, let's first understand what normal DNS traffic looks like.

Setup - Terminal 1 (DNS-DHCP VM):

1. Open a terminal and SSH into the DNS-DHCP VM:

```
cd DNS && vagrant ssh
```

2. Start capturing DNS traffic on port 53 with live display:

```
sudo tshark -i eth1 -f "port 53" -Y "dns.flags.response == 0" \
      -T fields -e dns.qry.name
```

3. Leave this running - you'll see DNS queries displayed live as they occur

Setup - Terminal 2 (Agent VM):

1. Open a second terminal and SSH into the Agent VM:

```
cd Agent && vagrant ssh
```

2. Send a few normal DNS queries using nslookup:

```
nslookup google.com
nslookup example.com
nslookup test.example.com
```

Analyzing Normal Traffic:

1. Observe the DNS queries appearing in Terminal 1 in real-time
2. Notice the simple, readable domain names (google.com, example.com, test.example.com)
3. Keep the tshark capture running for the next task

Question 2.1a: Count how many DNS query lines appeared in your tshark output. How many DNS queries were captured?

Question 2.1b: What do normal DNS subdomains look like? Are they human-readable?

3.2 Task 1.2: Capturing Covert Channel Traffic

Now let's capture traffic from the covert channel agent and compare it to normal DNS.

Setup - Terminal 1 (DNS-DHCP VM):

1. If your previous tshark capture is still running, stop it with Ctrl+C
2. Start a new live capture for covert traffic:

```
sudo tshark -i eth1 -f "port 53" -Y "dns.flags.response == 0" \
      -T fields -e dns.qry.name
```

Setup - Terminal 2 (Agent VM):

1. Navigate to the agent source directory:

```
cd src
```

2. Send a message using base32 encoding:

```
python3 main.py --channel base32 --data "Hello" --mode send
```

3. Send another message using hex encoding:

```
python3 main.py --channel hex --data "World" --mode send
```

4. Send one more message with base32:

```
python3 main.py --channel base32 --data "DNS" --mode send
```

Analyzing Covert Traffic:

1. Observe the covert DNS queries appearing live in Terminal 1
2. You should see encoded subdomains like "irhfg.example.com" (base32) and hex-encoded strings
3. Compare these subdomains to the normal ones from Task 1.1
4. Stop the capture with Ctrl+C when done

Question 2.2a: What differences do you notice between normal DNS queries and covert channel queries? How do the subdomains look different?

Question 2.2b: How many covert DNS query lines appeared in your tshark output?

3.3 Task 1.3: Identifying Encoding Types

DNS covert channels use different encoding schemes. The main ones used in this lab are:

- **Hexadecimal encoding**: Uses characters 0-9 and a-f (16 possible values)
- **Base32 encoding**: Uses characters a-z and 2-7 (32 possible values)
- **XOR + Base32 encoding**: Adds an encryption layer on top of base32 for additional obfuscation (bonus - see end of section)

Hints for identification:

- If subdomain contains **only digits (0-9) and letters a-f**, it's likely hexadecimal
- If subdomain contains **letters and digits 2-7** (but never 0, 1, 8, 9), it's likely base32

- Base32 never uses characters 0, 1, 8, or 9 to avoid confusion
- Character set size: hex (16) is less efficient than base32 (32)
- XOR-encrypted messages look like base32 but decode to garbage unless you have the encryption key

Question 2.3a: Imagine looking at a captured covert traffic where a subdomain starts with "ob4xa". Based on the character set hints above, what encoding do you think this is? List the unique characters you see and explain your reasoning.

3.4 Task 1.4: Decoding Covert Messages

Now that you've identified the encoding types, use the decoder tool to extract the hidden messages.

Steps:

1. In Terminal 2 (Agent VM), make sure you're in the source directory (src)
2. From your live tshark capture, you should have seen the subdomain "irhfg" (base32 encoded "DNS")
3. Decode it using the decoder:

```
python3 decode.py --channel base32 --encoded "irhfg"
```

4. You should see the decoded message

Expected output example:

```
======================================================================
DECODED MESSAGE
======================================================================

DNS

======================================================================

Decoded 3 bytes
  Text (UTF-8): DNS
  Hex: 444e53
  Bytes: b'DNS'
```

Question 2.4a: Now find and decode the "World" message from your tshark capture. First, extract the hex-encoded subdomain (remove the ".example.com" part), then decode it:

```
python3 decode.py --channel hex \
       --encoded "YOUR_HEX_SUBDOMAIN_HERE"
```

What is the decoded message? Paste both the encoded subdomain and decoded result.

Question 2.4b: Decode the "Hello" message. Extract the base32 subdomain from your tshark capture and decode it. What subdomain did you find, and what did it decode to?

3.5 Task 1.4c: Saving Captures to Pcap Files (For Submission)

While live captures are convenient for analysis, you may need to save traffic to a file for submission or later review. This task demonstrates how to capture to a pcap file and transfer it to your host machine.

Capturing to a Pcap File:

Setup - Terminal 1 (DNS-DHCP VM):

1. Start capturing to a file (without live display):

```
sudo tshark -i eth1 -f "port 53" -w /tmp/covert_evidence.pcap
```

2. You won't see output - packets are being saved to the file

Setup - Terminal 2 (Agent VM):

1. Send multiple covert messages:

```
python3 main.py --channel base32 --data "Evidence" --mode send
python3 main.py --channel hex --data "Lab" --mode send
python3 main.py --channel base32 --data "Capture" --mode send
```

Analyzing the Saved Pcap:

1. Return to Terminal 1 and stop the capture with Ctrl+C
2. Read the saved pcap file (filtering for requests only):

```
sudo tshark -r /tmp/covert_evidence.pcap \
      -Y "dns.flags.response == 0" \
      -T fields -e dns.qry.name
```

3. You should see the three encoded subdomains

Transferring the Pcap to Your Host Machine:

The pcap file is inside the VM. To submit it or analyze it on your host machine, you need to copy it out:

1. First, copy the file to the shared Vagrant directory (still in Terminal 1 - DNS-DHCP VM):

```
sudo cp /tmp/covert_evidence.pcap /vagrant/
sudo chmod 644 /vagrant/covert_evidence.pcap
```

2. Exit the VM:

```
exit
```

3. On your host machine, the file is now in your DNS directory:

```
# Navigate to your DNS directory (outside the VM)
cd DNS
ls -l covert_evidence.pcap
```

4. You can now open it with Wireshark, submit it, or analyze it further

Question 2.4c: Create a pcap file capturing your own covert messages. Save it as `task1_submission.pcap` and transfer it to your host machine. How many DNS query packets (requests only) are in your file? Use this command to count:

```
sudo tshark -r /tmp/task1_submission.pcap \
      -Y "dns.flags.response == 0" | wc -l
```

Note: Keep this file for your lab submission. Include both the pcap file and the decoded messages in your report.

3.6 Task 1.5: Generating Encoded Messages

Now that you understand how to decode messages, let's learn how to generate encoded messages without sending them.

Question 2.5a: You need to exfiltrate the message "SecretData" using base32 encoding. What would the subdomain look like?

Option 1 - Use the agent in print mode (doesn't send):

```
python3 main.py --channel base32 --data "SecretData" --mode print
```

The output will show you the encoded subdomain without actually sending it.

Option 2 - Send it and capture with tshark:

1. Start tshark on DNS-DHCP VM:

   ```
   sudo tshark -i eth1 -f "port 53" \
         -T fields -e dns.qry.name
   ```

2. Send the message from Agent VM:

   ```
   python3 main.py --channel base32 --data "SecretData" --mode send
   ```

3. Observe the subdomain in the tshark output

Question 2.5b: What is the base32-encoded subdomain for "SecretData"? Verify by decoding it - does it decode back correctly?

3.7 Task 1.6: Code Completion Exercise

The agent includes a training exercise to help you understand encoding. You'll complete a simple base32 encoding function.

Objective: Complete the `encode()` function in a training exercise file.

File location: `Agent/src/channels/base32_channel_exercise.py`

Note: Once you modify you local files, you need to reboot the Agent VM to load changes:

```
exit
vagrant reload
vagrant ssh
```

Testing your implementation:

```
cd src
python3 main.py --channel base32-exercise \
       --data "TestMessage" --mode print
```

You should see complete and encrypt the message correctly.

Question 2.6a: If you encode the same message, with "base32" instead of "base32-exercise", what subdomain do you get? Does it match your implementation?

Question 2.6b: After completing the function successfully, encode the message "Covert" using your function. What is the encoded output?

Question 2.6c: Verify your encoded "Covert" message by decoding it with the decoder tool. Does it decode back to "Covert" correctly? You can use mode `print` to avoid sending and just see the encoded output.

3.8 Task 1.7 (Bonus): XOR Encryption Layer

Optional Challenge: For advanced obfuscation, the lab includes an XOR-encrypted base32 channel that adds an encryption layer on top of base32 encoding. This makes the message much harder to decode without knowing the encryption key.

How XOR Encryption Works:

1. The message is first XOR-encrypted with a secret key
2. The encrypted bytes are then base32-encoded
3. Without the key, the decoded message appears as random garbage
4. With the correct key, the message can be decrypted

Trying XOR Encryption:

Step 1 - Send an encrypted message:

```
# Send a message with XOR encryption using key "Secret"
python3 main.py --channel xor --data "Hidden" \
       --key "Secret" --mode print
```

Step 2 - Try decoding without the key:

```
# Extract the subdomain from the output above
python3 decode.py --channel base32 --encoded "SUBDOMAIN_HERE"
```

You'll see garbage output because it's encrypted!

Step 3 - Decode with the correct key:

```
python3 decode.py --channel xor --encoded "SUBDOMAIN_HERE" \
       --key "Secret"
```

Now you should see "Hidden" correctly decoded.

Question 2.7 (Bonus):

a) Send a message using XOR encryption with your own secret key. What subdomain is generated?
b) Try to decode it without the key using base32. What do you see?
c) Decode it with the correct key. Does it work?
d) Why is XOR encryption useful for covert channels? What additional security does it provide?

4 Task 3: Multi-Packet Covert Channels

Estimated time: 20 minutes

Multi-packet covert channels encode data across multiple DNS queries by manipulating DNS header fields or query characteristics. In this task, you'll investigate channels that require multiple packets to transmit a single message.

4.1 Task 2.1: Understanding the RD Flag Channel

The Recursion Desired (RD) flag channel encodes data by toggling the RD flag bit in DNS query headers. This is extremely covert as RD flag variation appears normal.

How it works:

- Each DNS query encodes exactly 1 bit of data
- RD flag set (1) = bit value 1
- RD flag clear (0) = bit value 0
- 8 queries needed per byte (8 bits = 1 byte)

Example: Character 'H' (ASCII 0x48 = binary 01001000)

Bit position	1	2	3	4	5	6	7	8
Bit value	0	1	0	0	1	0	0	0
RD flag	Clear	Set	Clear	Clear	Set	Clear	Clear	Clear

Question 3.1: How many DNS queries are needed to transmit the 2-byte message "Hi"? Show your calculation.

Question 3.2: If you observe 64 DNS queries with varying RD flags, how many bytes of data were transmitted?

4.2 Task 2.2: Sending and Capturing RD Flag Messages

Now you'll send messages using the RD flag channel and capture them to see how multi-packet encoding works.

Setup - Terminal 1 (DNS-DHCP VM):

1. Start a new capture showing RD flags:

```
sudo tshark -i eth1 -f "port 53" -Y "dns.flags.response == 1" \
      -T fields -e frame.number -e dns.flags.recdesired \
      -e dns.qry.name
```

2. Leave this running to capture the RD flag values

Setup - Terminal 2 (Agent VM):

1. In the Agent VM, send a short message using RD flag channel:

```
python3 main.py --channel rd --data "Hi" --mode send
```

2. You should see output showing how many queries are needed (16 for "Hi")
3. Send another message:

```
python3 main.py --channel rd --data "OK" --mode send
```

Analysis:

1. Return to Terminal 1 and observe the RD flag values

2. You should see patterns like: 0,1,0,0,1,0,0,0,0,1,1,0,1,0,0,1
3. Stop the capture with Ctrl+C

Question 3.3: Looking at your tshark output, record the sequence of RD flag values for the "Hi" message (should be 16 values: 8 bits for 'H' + 8 bits for 'i'). Write them as comma-separated values.

4.3 Task 2.3: Decoding Your Multi-Packet Messages

The decoder tool can decode RD flag sequences back to the original message. Let's verify the messages you sent.

Steps:

1. In Terminal 2 (Agent VM), use the decoder to verify your "Hi" message:

```
python3 decode.py --channel rd \
       --encoded "RD:YOUR_SEQUENCE_FROM_QUESTION_2.3"
```

2. Replace `YOUR_SEQUENCE_FROM_QUESTION_2.3` with the actual RD flags you captured
3. Verify it decodes to "Hi"

Expected output example:

```
======================================================================
DECODED MESSAGE
======================================================================

Hi

======================================================================

Decoded 2 bytes
  Text (UTF-8): Hi
  Hex: 4869
  Bytes: b'Hi'
```

Question 3.4: Now decode the "OK" message using the RD flag sequence you captured. Paste the RD sequence you use and confirm it decodes correctly to "OK".

Question 3.4b: For verification, manually decode this RD flag sequence. What message was transmitted?

```
RD:0,1,0,0,0,0,0,1
```

Hint: This is an 8-bit (1-byte) message. What ASCII character is 01000001 in binary?

4.4 Task 2.4: Other Multi-Packet Channels

The lab includes several other multi-packet covert channels. Explore them using the decoder's help:

```
python3 decode.py --help
```

Available multi-packet channels:

- **ttl**: Encodes data in DNS TTL (Time To Live) values
- **qtype**: Encodes data by varying DNS query types (A, AAAA, MX, etc.)
- **txid**: Encodes data in transaction ID field

- **labels**: Encodes data in the number of domain labels
- **case**: Encodes data in case variations of domain names

Question 3.5: The TXID channel encodes data in the DNS transaction ID field (16 bits per query). How many queries are needed to transmit a 10-byte message? Show your calculation.

Hint: Each query has a 16-bit (2-byte) transaction ID.

4.5 Task 2.5: Testing Another Multi-Packet Channel

To verify you understand multi-packet encoding, test another channel: TXID (Transaction ID). This channel is more efficient than RD flag because it encodes 2 bytes (16 bits) per query instead of 1 bit.

Understanding TXID Channel:

- Each DNS query has a 16-bit Transaction ID field
- TXID channel encodes 2 bytes per query
- Much more efficient than RD flag (16x faster!)
- Example: "Hello" (5 bytes) = only 3 queries needed

Steps - Start a new capture (Terminal 1):

1. On DNS-DHCP VM, start capturing with TXID display:

```
sudo tshark -i eth1 -f "port 53" -Y "dns.flags.response == 1" \
      -T fields -e frame.number -e dns.id -e dns.qry.name
```

Steps - Send TXID message (Terminal 2):

1. On Agent VM, send a message using TXID channel:

```
python3 main.py --channel txid --data "Test" --mode send
```

2. Note how many queries are sent (should be 2 for "Test" = 4 bytes)
3. Stop the capture in Terminal 1

Question 3.6: Based on your capture, record the Transaction ID values you see. How many queries were needed for the 4-byte message "Test"? Does this match your expectation (4 bytes ÷ 2 bytes/query = 2 queries)?

Verifying Decoding: Use the decoder tool to decode the captured TXID values. Convert your HEX TXIDs in the capture into decimal base (https://www.rapidtables.com/convert/number/hex-to-decimal.html). Does it correctly reconstruct the original message "Test"? What TXID (decimal base) values did you use?

Question 3.7: Calculate: If you need to exfiltrate a 100-byte document, how many DNS queries would you need using:

- RD flag channel (1 bit per query)?
- TXID channel (16 bits per query)?

Show your calculations and compare the efficiency.

4.6 Task 2.6: Bonus - Complete Workflow Verification

Optional Challenge: Complete the full send-capture-decode workflow for one more channel.

Steps:

1. Choose a message (2-3 characters)

2. Choose a channel: ttl, qtype, or labels
3. Start a capture on DNS-DHCP VM (Terminal 1). Check documentation for channel-specific tshark filters.
4. Send the message from Agent VM (Terminal 2):

```
# Example with labels channel
python3 main.py --channel labels --data "AB" --mode both
```

5. The `--mode both` will show you what's being sent AND send it
6. Verify in your capture that you can see the encoded data
7. Use the decoder to decode it back

Question 3.8 (Bonus): Document your complete workflow:

- What message did you send?
- What channel did you use?
- How many packets were required?
- What did the encoded data look like in the capture?
- Did it decode correctly?