

DNS Security Lab

DIG - Cache Poisoning - DNSSEC



Client: 10.9.0.2

Attacker: 10.9.0.10

Resolver: 10.9.0.53

Authoritative: 10.9.0.153

Network Security – Introduction to Cybersecurity

Luca Campa



Academic use only. This lab contains intentionally vulnerable DNS behavior and must run only in isolated Docker networks.

Contents

Submission Rules & Setup	2
1 Overview	3
1.1 Environment	3
2 Start and Health Check	3
3 Task 1: Custom dig with Scapy	4
3.1 Background: DNS Packet Structure	4
3.2 Crafting a DNS Query with Scapy	4
3.3 Task Instructions	5
4 Task 2: Classic Cache Poisoning	5
5 Task 3: Enable DNSSEC (Step-by-Step)	6
5.1 Step 3.1: Create DNSSEC Keys on Authoritative	6
5.2 Step 3.2: Sign Zone	6
5.3 Step 3.3: Switch Authoritative to Signed Zone	6
5.4 Step 3.4: Enable Validation on Resolver	7
5.5 Step 3.5: Verify DNSSEC Protection	7
6 Troubleshooting	7

Rules & Setup

Rules

- Do not attack external DNS infrastructure or public resolvers.
- After running `docker compose up`, turn off any external network connections to ensure the lab environment is isolated. The provided docker network is already isolated, but better safe than sorry.
- You can use any online resources, but you must write the code yourself. I'm aware of the usage of AI tools, but I expect you to understand, describe and justify your solution.
- You can ask for help from the instructor or teaching assistants during office hours.
- Please submit your code and a brief report (no page limit) describing your implementation and results. Upload a zip file containing the report (a PDF file) and your scripts. Include a short guide on how to execute your scripts if instructions are different from the one presented in this document. The report should include:
 - A description of your implementation approach for each exercise.
 - An analysis of the results, including any challenges you faced and how you overcame them.

The report should NOT include:

- Your full scripts.
- If you want to include code snippets, make sure they are relevant to the discussion.

You can refer to the code by using the line numbers in the scripts.

The scripts must be well-commented. Each function should have a clear description of type of inputs, outputs and what it does. The code should be organized and readable.

- You will be asked to present one of your solutions (not all the exercises) during the Friday Lab Sessions. Make sure to be prepared to explain your code and the results you obtained.
- Keep the final state reproducible so the lab can be re-run from a clean container start.

Setup

What to do first

1. Create or start a VM for the lab.
2. Install the required packages inside the VM.
3. Start the Docker lab environment and verify the DNS containers are reachable.

```
sudo apt-get update
sudo apt-get install -y python3 python3-pip docker.io docker-
  compose-plugin dnsutils tcpdump tshark net-tools iputils-ping
  wireshark

pip3 install scapy dnspython cryptography

sudo systemctl enable docker
sudo systemctl start docker
sudo usermod -aG docker $USER
```

After logging back in, open the lab folder and start the exercises from the commands below.

1. Overview

This variant covers three stages:

1. Querying DNS with a custom Scapy client
2. Exploiting a vulnerable resolver with classic cache poisoning
3. Defending the setup with DNSSEC

Learning Objectives

- Understand DNS request/response fields using Scapy
- Perform cache poisoning against a weak resolver in a controlled setup
- Enable DNSSEC validation and observe why poisoning fails afterward

1.1 Environment

Container	Role	IP	Notes
client	DNS client	10.9.0.2	Legitimate requests
attacker	Attack node	10.9.0.10	Runs attack scripts
resolver	Vulnerable BIND9 resolver	10.9.0.53	Fixed source port 33333
authoritative	example.com zone server	10.9.0.153	DNSSEC signer

Resolver weaknesses used in the lab:

1. Fixed upstream source port (33333)
2. DNSSEC validation initially disabled

To analyze the traffic passing through the docker network, you can either use tcpdump or Wireshark on the host machine. Make sure to capture on the correct Docker network interface (e.g., br-xxxxxx) and apply filters for the relevant packets (e.g. udp).

2. Start and Health Check

```
cd dns-lab/no_kaminsky
docker compose up -d --build
docker compose ps
```

Optional sanity check:

```
docker exec client dig @10.9.0.53 www.example.com +short
```

Expected answer before attacks: 1.2.3.4

3. Task 1: Custom dig with Scapy

3.1 Background: DNS Packet Structure

A DNS message has a fixed-size 12-byte header followed by variable-length sections:

DNS Header (12 bytes)							
Transaction ID (16 bits)							
QR	Opcode	AA	TC	RD	RA	Z	RCODE
QDCOUNT (16 bits)							
ANCOUNT (16 bits)							
NSCOUNT (16 bits)							
ARCOUNT (16 bits)							

Key header flags:

- **QR**: 0 = Query, 1 = Response
- **RD** (Recursion Desired): ask the resolver to recurse on our behalf
- **RA** (Recursion Available): set by resolvers that support recursion
- **AA** (Authoritative Answer): set by authoritative name servers
- **TC** (Truncated): response was truncated; retry over TCP

3.2 Crafting a DNS Query with Scapy

Below is a snippet showing how to build a DNS query packet using Scapy. This is just an example, you will have to modify it and also understand how to parse it to extract the relevant fields from the response.

```

1 txid = random.randint(0, 65535)
2
3 # DNS layer: QR=0 (query), RD=1 (recursion desired)
4 dns_layer = DNS(
5     id=txid,
6     rd=1,                # recursion desired
7     qd=DNSQR(
8         qname=domain,
9         qtype=qtype,     # "A", "AAAA", "MX", "TXT", "NS", ...
10        qclass="IN",
11        # ....
12    )
13 )
14
15 if use_tcp:
16     # TCP DNS: prepend a 2-byte length field (handled by Scapy
17     # internally)
18     pkt = IP(dst=DNS_SERVER) / TCP(dport=53, sport=random.randint
19         (1024,65535), flags="S") / dns_layer
20 else:
21     pkt = IP(dst=DNS_SERVER) / UDP(dport=53, sport=random.randint
22         (1024,65535)) / dns_layer

```

Refer to <https://scapy.readthedocs.io/en/latest/introduction.html> for more details on how to use Scapy to build and parse packets.

3.3 Task Instructions

>_ Task 1

Goal: understand DNS request/response fields by sending a custom query packet.

Create your own `my_dig.py` script (using Scapy), then:

```
docker cp solutions/my_dig.py attacker:/solutions/my_dig.py
docker exec -it attacker /bin/bash
cd /solutions
python3 my_dig.py www.example.com 10.9.0.53 <QUERY_TYPE>
```

Note that 10.9.0.53 is the resolver's IP. You can specify the query type (e.g., A, AAAA, NS, TXT) as an optional third argument. If not specified it defaults to A. The script should print the DNS question and answer sections from the response, allowing you to analyze the fields and understand how DNS queries and responses work.

4. Task 2: Classic Cache Poisoning

>_ Task 2

Goal: poison resolver cache so `www.example.com` resolves to `6.6.6.6`.

This attack relies on the resolver's predictable behavior (fixed source port, no DNSSEC) and must only be run in this controlled lab. The MITM position is obtained with ARP spoofing: the attacker sends forged ARP replies so the resolver maps the gateway/authoritative peer IP to the attacker's MAC address (and vice versa), causing DNS traffic to pass through the attacker first. From that position, the attacker races the legitimate reply by sending a forged DNS response.

At Layer 3/4, the forged packet must look exactly like it came from the expected upstream server. That means spoofing the source IP as the authoritative server, setting destination IP to the resolver, matching source and destination UDP ports (including the resolver's fixed destination port), and guessing the correct DNS Transaction ID. If these fields match what the resolver is waiting for, the forged answer can be accepted and cached before the real one arrives. To sum up, the attacker must ensure to match the following fields:

- Source IP
- Destination IP
- Source Port
- Destination Port
- Transaction ID

In `solutions/cache_poisoning.py` you will find a skeleton of the attack script. You need to fill in the missing parts to perform the poisoning attack successfully. Then:

1. Flush resolver cache:

```
docker exec resolver rndc flush
```

2. Start attack script:

```
docker cp solutions/cache_poisoning.py attacker:/solutions/
cache_poisoning.py
docker exec -it attacker bash
cd /solutions
python3 cache_poisoning.py
```

3. Trigger resolution and verify cache content:

```
docker exec client dig @10.9.0.53 www.example.com +short
```

Expected poisoned answer: 6.6.6.6

🔧 5. Task 3: Enable DNSSEC (Step-by-Step)

> Task 3

Goal: sign the authoritative zone and enable resolver validation so poisoning fails.

5.1 Step 3.1: Create DNSSEC Keys on Authoritative

```
docker exec -it authoritative /bin/bash
cd /etc/bind

dnssec-keygen -a RSASHA256 -b 1024 -n ZONE example.com
dnssec-keygen -a RSASHA256 -b 2048 -n ZONE -f KSK example.com
cat Kexample.com.*.key >> example.com.zone
```

5.2 Step 3.2: Sign Zone

```
dnssec-signzone -A -3 $(head -c 1000 /dev/urandom | shasum | cut -b
1-16) -N INCREMENT -o example.com -t example.com.zone
```

This produces `example.com.zone.signed`.

5.3 Step 3.3: Switch Authoritative to Signed Zone

Edit `/etc/bind/named.conf.local` inside the authoritative container:

1. Change file `"/etc/bind/example.com.zone"`;
2. To file `"/etc/bind/example.com.zone.signed"`;

Restart authoritative (the following command is run inside the authoritative container):

```
service named restart
```

You can also use `docker cp` and `docker exec` to edit the file from outside the container if you prefer.

5.4 Step 3.4: Enable Validation on Resolver

```
docker exec -it resolver bash
nano /etc/bind/named.conf.options
```

Change:

1. `dnssec-validation no;`
2. To `dnssec-validation auto;`

Print KSK DNSKEY:

```
docker exec authoritative sh -lc "grep -h 'DNSKEY 257 3 8' /etc/bind/
Kexample.com*.key"
```

Add trust anchor in resolver `named.conf.options`:

```
trust-anchors {
    "example.com." static-key 257 3 8 "<BASE64_FROM_KSK>";
};
```

Restart resolver:

```
docker exec resolver service named restart
```

5.5 Step 3.5: Verify DNSSEC Protection

```
docker exec client dig @10.9.0.53 www.example.com +dnssec
```

Repeat Task 2. Expected outcome: poisoning no longer persists.

6. Troubleshooting

```
docker exec resolver rndc flush
docker compose up -d --force-recreate resolver authoritative
docker logs resolver --tail 100
docker logs authoritative --tail 100
```

Seeing `SERVER: 127.0.0.11` in container `dig` output can be normal with Docker embedded DNS forwarding.