Лабораторна робота № 11(3.2.9)
Створення пакетів за допомогою Scapy
Хід роботи:

Частина 1: Дослідження інструменту Scapy.

Крок 1: Дослідження документації та ресурсів Scapy

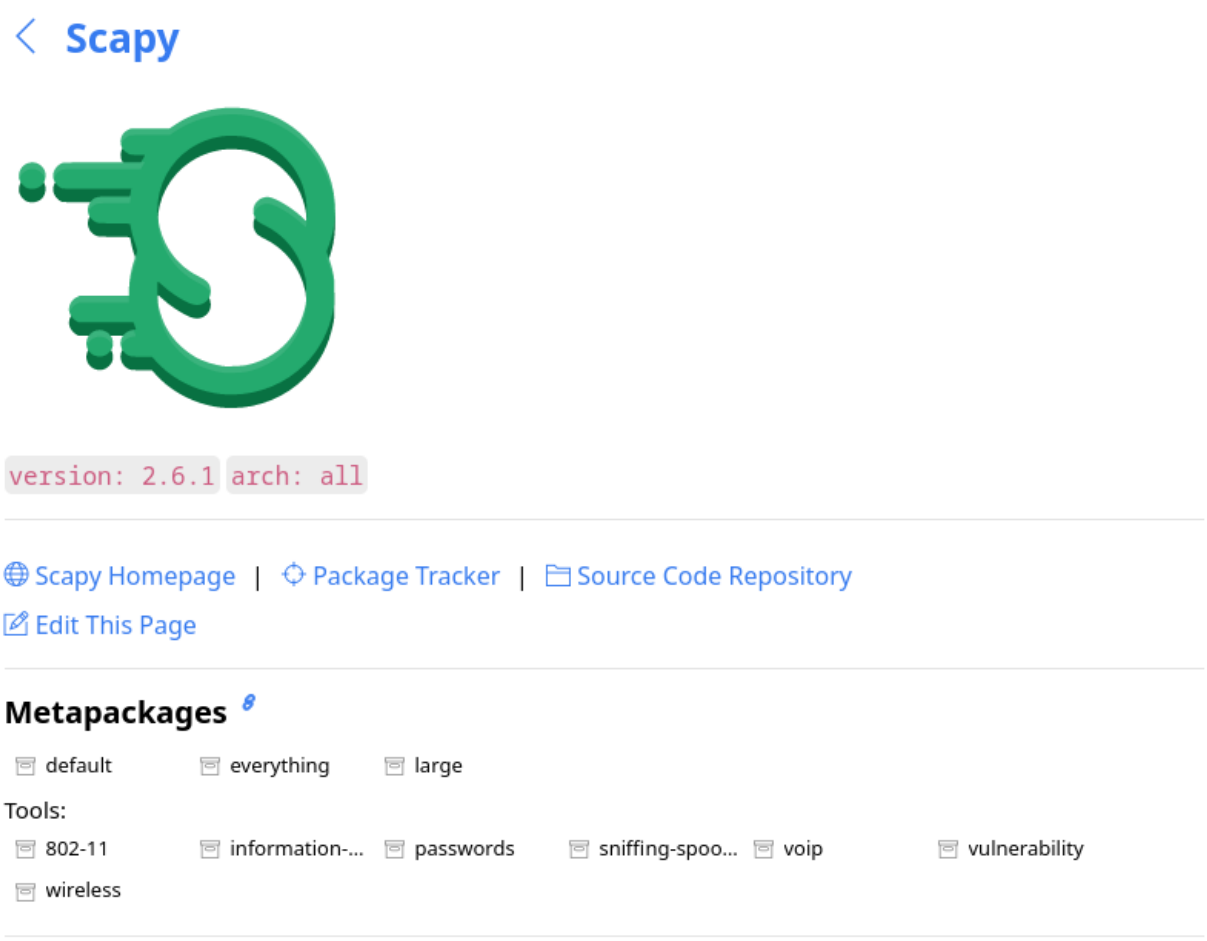

Рис. 1. Головна сторінка документації Scapy на веб-сайті

Питання: Як автор описує можливості Scapy в першому абзаці сторінки?

Відповідь: Автор Philippe Biondi описує Scapy як потужний інтерактивний інструмент для маніпуляції пакетами, який може підробляти або декодувати пакети великої кількості протоколів, відправляти їх по мережі, перехоплювати, зіставляти запити та відповіді та багато іншого. Scapy може легко обробляти більшість класичних завдань, таких як сканування, трасування маршруту, зондування, атаки або виявлення мережі. Він може замінити hping, nmap, arpspoof, arp-sk, arping, tcpdump, tethereal, p0f тощо, і виконувати більшість завдань, які ці інструменти не можуть виконати, як-то відправлення недійсних фреймів, впровадження власних 802.11 фреймів, комбінування технік тощо.

Крок 2: Використання інтерактивного командного режиму Scapy

a-b. Запуск Scapy з правами root:

sudo su

scapy

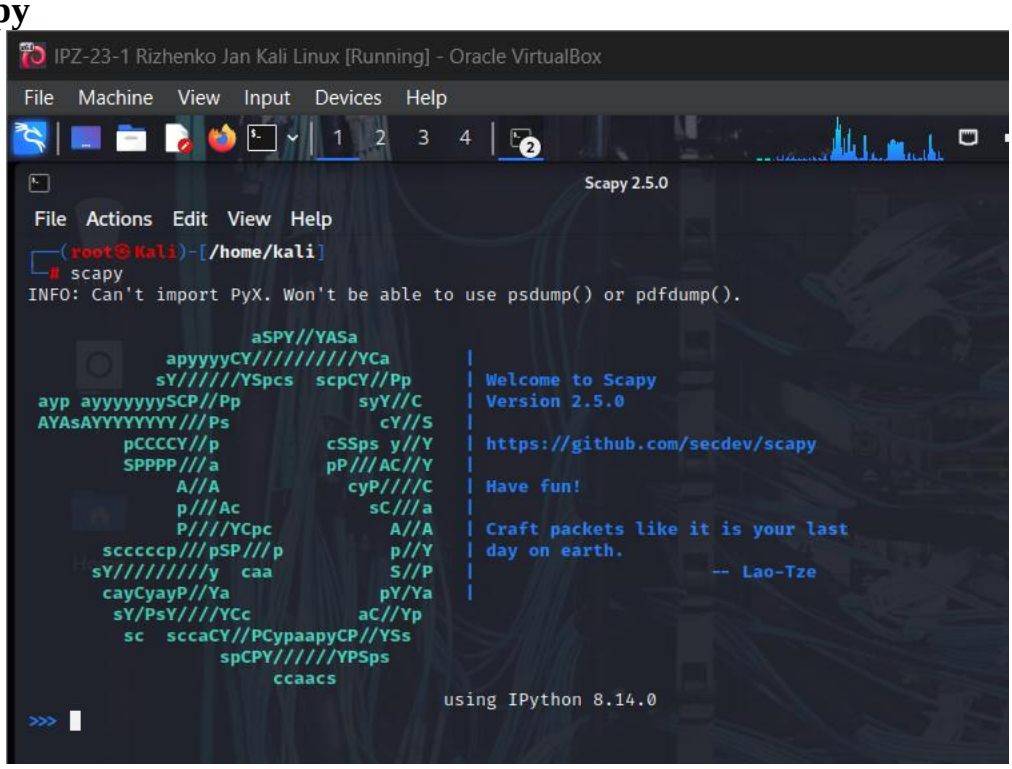

Рис. 2. Запуск Scapy в інтерактивному режимі

c. Перегляд доступних протоколів:

ls()

Рис. 3. Список доступних форматів пакетів та протоколів у Scapy

Питання: Скільки типів форматів пакетів TFTP перераховано?
Відповідь: У Scapy перераховано 5 типів форматів пакетів TFTP:

- TFTP - базовий клас TFTP
- TFTP_RRQ - Read Request (запит на читання)
- TFTP_WRQ - Write Request (запит на запис)
- TFTP_DATA - Data packet (пакет даних)
- TFTP_ACK - Acknowledgment (підтвердження)
- TFTP_ERROR - Error packet (пакет помилки)

(Фактична кількість може становити 5-6 залежно від версії Scapy)
Крок 3: Дослідження полів у заголовку пакета IPv4
b. Перегляд полів IP пакета:

ls(IP)

```
>>> ls(IP)
version    : BitField  (4 bits)                  = ('4')
ihl        : BitField  (4 bits)                  = ('None')
tos        : XByteField                          = ('0')
len        : ShortField                          = ('None')
id         : ShortField                          = ('1')
flags      : FlagsField                          = ('<Flag 0 ()>')
frag       : BitField  (13 bits)                 = ('0')
ttl        : ByteField                           = ('64')
proto      : ByteEnumField                       = ('0')
chksum     : XShortField                         = ('None')
src        : SourceIPField                       = ('None')
dst        : DestIPField                         = ('None')
options    : PacketListField                     = ('[]')
>>>
```

Рис. 4. Детальний список полів заголовка IP пакета в Scapy

Питання: Чи є відмінності між полями в деталях IP у Scapy та заголовком пакета, описаним у кроці 3а?
Відповідь: Так, існують деякі відмінності в термінології та представленні:

- Scapy використовує скорочення "ihl" (Internet Header Length), тоді як стандартний опис може використовувати повну назву
- Поле "tos" (Type of Service) у Scapy відповідає полю "Differentiated Services (DS)" у стандартному описі
- Scapy показує типи полів (BitField, ByteField, ShortField), чого немає в стандартному описі заголовка
- Значення за замовчуванням відображаються в Scapy, що полегшує створення пакетів
- Загалом структура та призначення полів залишаються однаковими, але Scapy надає більш програмістсько-орієнтоване представлення

Питання: Яке поле ви б змінили, щоб створити пакет, який згенерує відповідь на цільову машину, а не на машину, яка фактично відправила пакет?
Відповідь: Я б змінив поле "src" (Source IP Address) - вихідну IP-адресу. Якщо підробити (spoofing) це поле та встановити адресу цільової машини або іншого

хоста, відповідь буде надіслана на підроблену адресу замість реальної машини-відправника. Це техніка, відома як IP spoofing, і вона часто використовується в атаках типу DDoS (наприклад, Smurf attack), коли зловмисник підробляє вихідну адресу, щоб спрямувати відповіді на жертву.

Частина 2: Використання Scapy для перехоплення мережевого трафіку

Крок 4: Використання функції sniff()

a-c. Захоплення трафіку:

sniff()

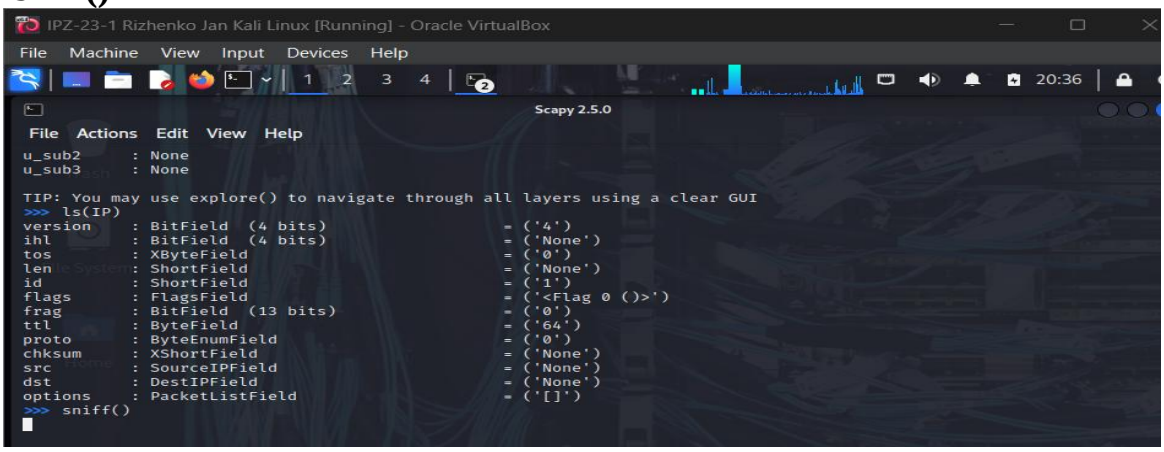

Рис. 5. Запуск захоплення пакетів за допомогою функції sniff()

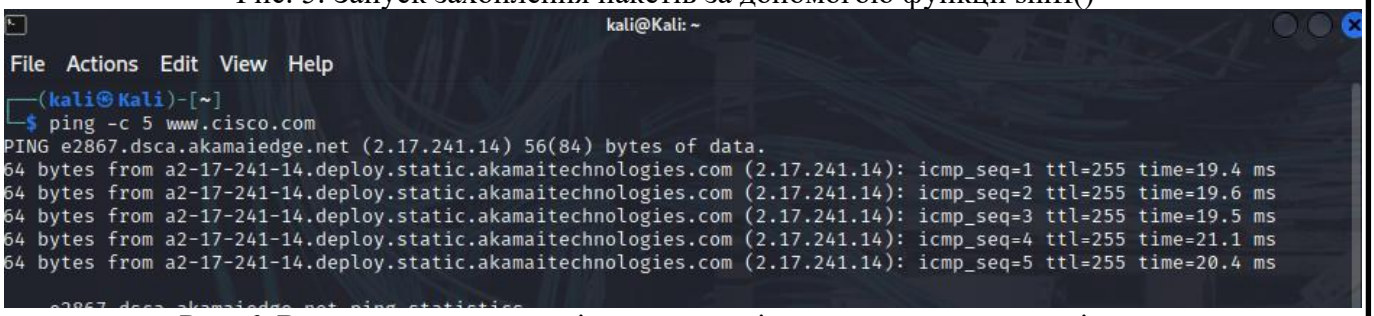

Рис. 6. Виконання команди ping до www.cisco.com з другого терміналу

d. Перегляд захопленого трафіку:

a=_

a.summary()

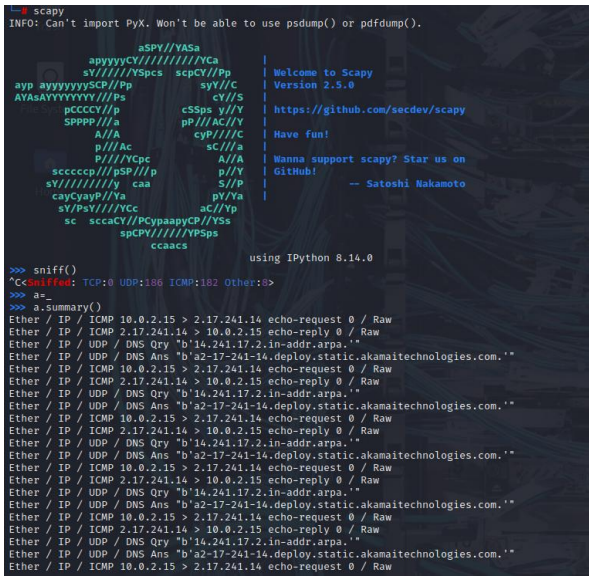

Рис. 7. Зведена інформація про захоплені пакети

Крок 5: Захоплення та збереження трафіку на конкретному інтерфейсі

a. Визначення інтерфейсу:

ifconfig

```
┌──(kali㉿Kali)-[~]
└─$ ifconfig
br-339414195aeb: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.5.5.1  netmask 255.255.255.0  broadcast 10.5.5.255
        inet6 fe80::42:10ff:fe6e:24bb  prefixlen 64  scopeid 0x20<link>
        ether 02:42:10:6e:24:bb  txqueuelen 0  (Ethernet)
        RX packets 75  bytes 4276 (4.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 23  bytes 3036 (2.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-355ee7945a88: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.1  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::42:94ff:fe8f:2e9  prefixlen 64  scopeid 0x20<link>
        ether 02:42:94:8f:02:e9  txqueuelen 0  (Ethernet)
        RX packets 72  bytes 10593 (10.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 15  bytes 2360 (2.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-internal: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

Рис. 8. Визначення інтерфейсу з адресою 10.6.6.1 (br-internal)

b-c. Захоплення трафіку на внутрішньому інтерфейсі:

sniff(iface="br-internal")

```
>>> sniff(iface="br-internal")
^C<Sniffed: TCP:168 UDP:0 ICMP:0 Other:0>
```

Рис. 9. Захоплення трафіку під час навігації до http://10.6.6.23

d. Перегляд результатів:

a=_

a.summary()

```
>>> sniff(iface="br-internal")
^C<Sniffed: TCP:168 UDP:0 ICMP:0 Other:0>
>>> a=_
>>> a.summary()
Ether / IP / TCP 10.6.6.1:34986 > 10.6.6.23:https S
Ether / IP / TCP 10.6.6.23:https > 10.6.6.1:34986 RA
Ether / IP / TCP 10.6.6.1:42276 > 10.6.6.23:http S
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:42276 SA
Ether / IP / TCP 10.6.6.1:42276 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.1:42276 > 10.6.6.23:http PA / Raw
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:42276 A
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:42276 PA / Raw
Ether / IP / TCP 10.6.6.1:42276 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:42276 PA / Raw
Ether / IP / TCP 10.6.6.1:42276 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:42276 PA / Raw
Ether / IP / TCP 10.6.6.1:42276 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.1:42276 > 10.6.6.23:http PA / Raw
Ether / IP / TCP 10.6.6.1:42282 > 10.6.6.23:http S
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:42282 SA
Ether / IP / TCP 10.6.6.1:42282 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:42276 PA / Raw
```

Рис. 10. Зведена інформація про захоплений HTTP трафік

Крок 6: Дослідження зібраних пакетів

a-b. Захоплення ICMP пакетів:

sniff(iface="br-internal", filter="icmp", count=10)

```
>>> sniff(iface="br-internal", filter="icmp", count=10)
<Sniffed: TCP:0 UDP:0 ICMP:10 Other:0>
```

Рис. 11. Захоплення 10 ICMP пакетів

c. Перегляд із номерами рядків:

a=_

a.nsummary()

```
>>> a.nsummary()
0000 Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
0001 Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
0002 Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
0003 Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
0004 Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
0005 Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
0006 Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
0007 Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
0008 Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
0009 Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
>>>
```

Рис. 12. Нумероване зведення захоплених ICMP пакетів

Питання: Який трафік відображається у виводі функції nsummary()?

Відповідь: У виводі функції nsummary() відображаються ICMP Echo Request та ICMP Echo Reply пакети між хостом 10.6.6.1 (Kali Linux) та 10.6.6.23 (цільовий хост). Зокрема:

- Парні номери (0, 2, 4, 6, 8) - ICMP Echo Request пакети від 10.6.6.1 до 10.6.6.23
- Непарні номери (1, 3, 5, 7, 9) - ICMP Echo Reply пакети від 10.6.6.23 до 10.6.6.1

Це стандартний обмін пакетами команди ping.

d. Перегляд деталей конкретного пакета:

a[2]

```
>>> a[2]
<Ether  dst=02:42:0a:06:06:17 src=02:42:f9:a2:e9:cb type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=84 id=59617 flags=DF
 frag=0 ttl=64 proto=icmp chksum=0x31a4 src=10.6.6.1 dst=10.6.6.23 |<ICMP  type=echo-request code=0 chksum=0xf52a id
=0x9c5a seq=0x2 unused='' |<Raw  load='ViDi\x00\x00\x00\x00\t\\xd3\x03\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x
16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567' |>>>>
>>>
```

Рис. 13. Детальна інформація про третій пакет у захопленні

Питання: Чому є два набори полів джерела та призначення?

Відповідь: Два набори полів джерела (src) та призначення (dst) присутні тому, що пакет складається з кількох рівнів інкапсуляції (моделі OSI/TCP IP):

Перший набір (Ethernet layer - канальний рівень): MAC-адреси джерела та призначення

- src: MAC-адреса відправника
- dst: MAC-адреса одержувача (або шлюзу)

Другий набір (IP layer - мережевий рівень): IP-адреси джерела та призначення

- src: IP-адреса відправника (10.6.6.1)
- dst: IP-адреса одержувача (10.6.6.23)

Це демонструє інкапсуляцію протоколів: IP пакет інкапсульовано в Ethernet фрейм. Кожен рівень має власні адреси для маршрутизації на відповідному рівні мережевої моделі.

e-g. Збереження захоплених даних:

wrpcap("capture1.pcap", a)

```
>>> wrpcap("capture1.pcap",a)
```

Рис. 14. Збереження захоплених пакетів у файл .pcap

```
┌──(kali㉿Kali)-[~]
└─$ ls
Desktop     Templates
Documents   Videos
Downloads   capture1.pcap
Music       'ides. It also tries to determine what operating system is running on each host that is up'
OTHER       kali_folder2
Pictures    test_file.txt
Public
```

Рис. 15. Перевірка створення файлу capture1.pcap

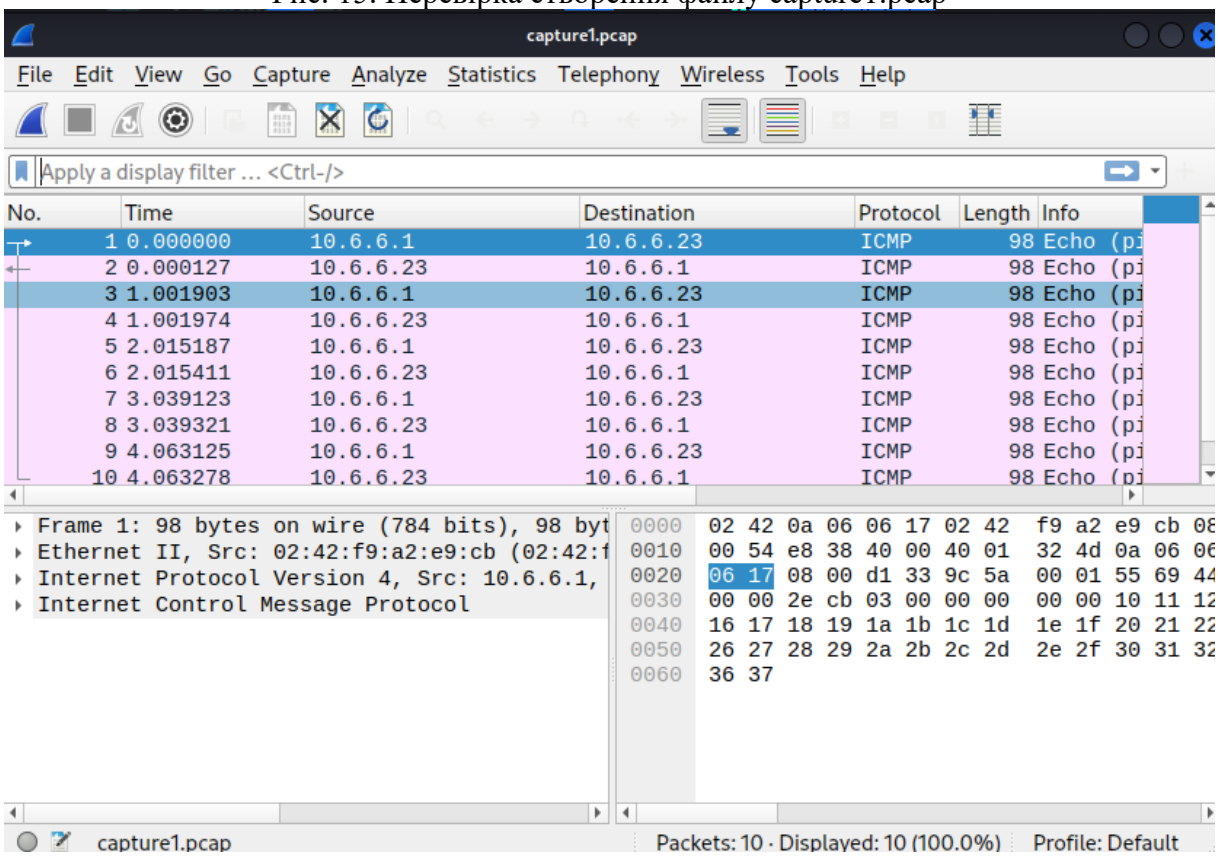

Рис. 16. Відкриття файлу capture1.pcap у Wireshark

Частина 3: Створення та відправлення ICMP пакета

Крок 1: Використання інтерактивного режиму для створення та відправлення власного ICMP пакета

a. Запуск захоплення в першому терміналі:

python>>> sniff(iface="br-internal")

b. Відправлення власного ICMP пакета з другого терміналу:

send(IP(dst="10.6.6.23")/ICMP()/"This is a test")

```
>>> send(IP(dst="10.6.6.23")/ICMP()/"This is a test")
.
Sent 1 packets.
```

Рис. 17. Відправлення власного ICMP пакета з модифікованим payload

c-d. Перегляд результатів:

a=_

a.nsummary()

Рис. 18. Зведення захоплених пакетів після відправлення власного ICMP

Питання: Які типи пакетів показані у виводі зведення?

Відповідь: У виводі зведення показані два типи ICMP пакетів:

ICMP Echo Request (тип 8) - відправлений від 10.6.6.1 до 10.6.6.23 з власним payload "This is a test"

ICMP Echo Reply (тип 0) - відповідь від 10.6.6.23 до 10.6.6.1, яка відображає той самий payload

Крок 2: Перегляд та порівняння вмісту ICMP пакетів

a[0]

a[1]

Рис. 19. Детальний вміст власного ICMP Echo Request пакета

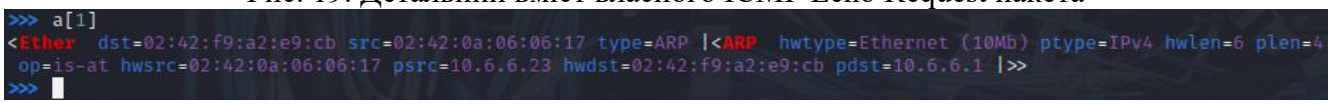

Рис. 20. Детальний вміст ICMP Echo Reply пакета

Питання: Що відрізняється між оригінальною ICMP розмовою та власною ICMP розмовою?

Відповідь: Основні відмінності між оригінальною та власною ICMP розмовою:

Payload (корисне навантаження):

- Оригінальний ping: стандартне корисне навантаження (зазвичай випадкові дані або послідовність символів)
- Власний пакет: містить текст "This is a test"

Розмір пакета:

- Оригінальний: стандартний розмір (зазвичай 64 байти)
- Власний: відрізняється залежно від довжини payload

ICMP ідентифікатор та послідовний номер:

Оригінальний: автоматично генеровані системою значення

Власний: значення за замовчуванням або нуль, якщо не вказано

Контрольна сума (checksum):

- Обидва мають правильну контрольну суму, але значення відрізняються через різний вміст

Це демонструє, що Scapy дозволяє повністю контролювати вміст пакетів, включаючи payload.

Частина 4: Створення та відправлення TCP SYN пакета

Крок 1: Запуск захоплення пакетів на внутрішньому інтерфейсі
a. Запуск захоплення:

sniff(iface="br-internal")

b. Відправлення TCP SYN пакета:

send(IP(dst="10.6.6.23")/TCP(dport=445, flags="S"))

```
>>> send(IP(dst="10.6.6.23")/TCP(dport=445,flags="S"))
.
Sent 1 packets.
```

Рис. 21. Відправлення TCP SYN пакета на порт 445

Крок 2: Перегляд захоплених пакетів
a-b. Аналіз результатів:

a=_

a.nsummary()

a[1]

```
>>> a.nsummary()
0000 Ether / ARP who has 10.6.6.23 says 10.6.6.1
0001 Ether / ARP is at 02:42:0a:06:06:17 says 10.6.6.23
0002 Ether / IP / TCP 10.6.6.1:ftp_data > 10.6.6.23:microsoft_ds S
0003 Ether / IP / TCP 10.6.6.23:microsoft_ds > 10.6.6.1:ftp_data SA
0004 Ether / IP / TCP 10.6.6.1:ftp_data > 10.6.6.23:microsoft_ds R
>>> a[1]
```

Рис. 22. Зведення захоплених TCP пакетів

```
>>> a[1]
<Ether  dst=02:42:f9:a2:e9:cb src=02:42:0a:06:06:17 type=ARP |<ARP  hwtype=Ethernet (10Mb) ptype=IPv4 hwlen=6 plen=4
 op=is-at hwsrc=02:42:0a:06:06:17 psrc=10.6.6.23 hwdst=02:42:f9:a2:e9:cb pdst=10.6.6.1 |>>
```

Рис. 23. Детальна інформація про відповідь від цільового хоста

Питання: Що вказує прапорець SA у пакеті, повернутому від 10.6.6.23?
Відповідь: Прапорець SA (SYN-ACK) у відповідному пакеті вказує на те, що:

- Порт 445 ВІДКРИТИЙ на цільовому хості 10.6.6.23
- Хост готовий встановити TCP з'єднання
- SYN-ACK є другим кроком у триетапному рукостисканні TCP (three-way handshake):

Крок 1: Клієнт надсилає SYN
Крок 2: Сервер відповідає SYN-ACK (підтверджує запит і пропонує власний SYN)
Крок 3: Клієнт відповідає ACK (не виконано в цьому прикладі)
Якби порт був закритий, хост відповів би пакетом RST (Reset). Якби порт був фільтрований firewall, відповіді не було б взагалі. Отримання SA-прапорця підтверджує, що служба SMB/CIFS активна та доступна на порту 445.

Питання для рефлексії
1. Як створення різних TCP SYN пакетів може використовуватися для проведення пасивної розвідки на цільовому хості?
Відповідь:
Створення TCP SYN пакетів є основою техніки SYN сканування (також відомого як "stealth scanning" або напівпідключення), яка використовується для виявлення

відкритих портів без встановлення повного TCP з'єднання. Це дозволяє проводити розвідку наступним чином:

Методи використання:

Сканування портів: Надсилання SYN пакетів на різні порти для визначення, які сервіси працюють

Визначення стану портів на основі відповідей:

- SA (SYN-ACK) = порт відкритий
- RST (Reset) = порт закритий
- Немає відповіді = порт фільтрується firewall

Fingerprinting ОС: Аналіз специфічних характеристик відповідей (TTL, розмір вікна, опції TCP)

Виявлення firewall та IDS: Тестування різних комбінацій прапорців для обходу систем безпеки

Картографування мережі: Визначення топології та захисних механізмів

Переваги для розвідки:

- Менш помітно, ніж повне TCP підключення (не завершує handshake)
- Швидше за традиційне сканування
- Часто не реєструється в логах додатків (тільки на рівні firewall)
- Дозволяє маніпулювати полями для обходу фільтрів

Технічні можливості:

- Зміна TTL для імітації різних відстаней
- Підроблення вихідної адреси для анонімності
- Налаштування розміру вікна та опцій TCP для fingerprinting
- Використання різних послідовностей портів для уникнення виявлення

Однак варто зазначити, що це насправді активна розвідка, а не пасивна, оскільки включає відправлення пакетів до цільового хоста.

2. Як створення ICMP echo-request пакета з підробленою вихідною адресою може створити атаку відмови в обслуговуванні (DoS) проти цільового хоста?

Відповідь:

Створення ICMP echo-request з підробленою вихідною адресою є основою Smurf Attack - класичної DDoS атаки з ампліфікацією. Механізм атаки працює наступним чином:

Принцип роботи Smurf Attack:

- Підроблення адреси: Зловмисник створює ICMP echo-request пакети, але встановлює поле src (вихідну IP-адресу) рівним IP-адресі жертви
- Широкомовне розсилання: Ці пакети відправляються на broadcast адресу мережі (наприклад, 192.168.1.255)
- Ампліфікація: Всі хости в цільовій мережі отримують broadcast і відповідають ICMP echo-reply
- Перевантаження жертви: Всі відповіді надсилаються до підробленої адреси (жертви), перевантажуючи її канал та ресурси

Чому це ефективна DoS атака:

- Ампліфікація трафіку: Один запит генерує десятки/сотні відповідей

- Анонімність: Справжній зловмисник залишається прихованим
- Споживання ресурсів: Жертва отримує величезну кількість трафіку
- Перевантаження мережі: Bandwidth жертви вичерпується
- Виснаження CPU: Обробка великої кількості пакетів

Варіації атаки:

- Fraggle Attack: Використання UDP замість ICMP
- DNS Amplification: Використання DNS серверів для ампліфікації
- NTP Amplification: Експлуатація NTP протоколу

Сучасний захист:

- Більшість мереж блокують directed broadcasts
- Ingress/egress filtering запобігає IP spoofing
- Rate limiting на ICMP трафікСучасні firewall та IPS системи виявляють такі патерни

Ця атака демонструє важливість валідації вихідних адрес та необхідність налаштування мережевих пристроїв для запобігання IP spoofing.

Висновок: У ході виконання лабораторної роботи було досліджено можливості Scapy як потужного Python-based інструменту для маніпуляції та створення мережевих пакетів. Освоєно інтерактивний режим роботи Scapy та вивчено структуру заголовків протоколів IP, ICMP та TCP. Практично застосовано функції sniff() для перехоплення мережевого трафіку на різних інтерфейсах з використанням фільтрів протоколів та обмеженням кількості пакетів. Виконано створення та відправлення власних ICMP пакетів з модифікованим payload, що продемонструвало можливості Scapy для повного контролю над вмістом пакетів. Реалізовано техніку SYN сканування шляхом створення та відправлення TCP SYN пакетів для визначення стану портів на цільовому хості, отримавши SYN-ACK відповідь, яка підтвердила, що порт 445 відкритий. Робота підкреслила важливість розуміння структури мережевих протоколів та продемонструвала, як інструменти створення пакетів можуть використовуватися як для легітимного тестування безпеки та розвідки, так і для потенційних зловмисних дій, включаючи DoS атаки через IP spoofing, що підкреслює необхідність належних заходів безпеки в мережевій інфраструктурі.