## Лабораторная работа 5. Перехват сетевых пакетов

Реализовать программу (сниффер) для перехвата пакетов в сети.

Реализовать удобный вывод данных о перехваченном пакете. Подсветка.

Любым программам для мониторинга сети, инструментам безопасности необходим перехват сетевых пакетов. На языке C# и для Фреймворка Net существует по крайней мере две библиотеки: SharPcap и WinPKFilter. Так же есть реализации на других языках.

Можно использовать RAW сокеты и самостоятельно анализировать, получаемые данные. Большинство этих библиотек требуют администраторских прав. Кроме того, представленные ниже примеры могут устареть, в связи с обновлением библиотек, поэтому желательно рассмотреть примеры, представленные в официальной документации.

Так как, большинство библиотек используют RAW сокеты, то ваше приложение для запуска потребует root прав или запуска под sudo, если вас не устраивают подобные обстоятельства, то можно установить виртуальную машину или виртуальную машину с контейнерной виртуализацией, например, Docker.

Далее приведен пример использования RAW сокетов и библиотеки scapy. Сначала рассмотрим RAW сокеты.

**5.1 Пример использования библиотеки RAW сокетов.**

Для доступа к низкоуровневым сокетам требуются root права, потому в виртуальному машине нужно будет запускать установку приложений pip через sudo, как и интерпретатор Python. Для упрощения работы создадим файл python с нашим приложением, поместив туда начальный код:

import socket

# 0x800 номер перехватываемого протокола при получении кадров Ethernet, в данном случае будут ловиться кадры содержащие пакеты IP.

rawSocket = socket.socket(socket.PF\_PACKET, socket.SOCK\_RAW, socket.htons(0x0800))

# получаем один кадр

data, addr = rawSocket.recvfrom(2048)

print(addr)

print(data)

Создадим shell файл со следующими командами. Здесь ! – имя процесса запущенного в фоновом режиме через команду &. Второй процесс сейчас не обязательно уничтожать, так как он итак прекращается, но на будущее если это будет снифер, мы можем оставить эту команду.

sudo python3 snif\_sr.py & APP\_ID1=$!

ping 127.0.0.1 & APP\_ID2=$!

sleep 5

kill -TERM $APP\_ID2

kill -TERM $APP\_ID1

Теперь, нужно дополнить наше простое приложение разбором полученных данных. Для этого подойдет модуль struct.

Попробуйте данный пример.

import socket

import struct

import binascii

rawSocket = socket.socket(socket.PF\_PACKET, socket.SOCK\_RAW, socket.htons(0x0800))

data, addr = rawSocket.recvfrom(2048)

# считываем заголовок Ethernet пакета

eth\_hdr = struct.unpack("!6s6s2s", data[0:14]) # 6 dest MAC, 6 host MAC, 2 ethType

# преобразуем MAC в 16-иричный формат

print(eth\_hdr)

print(binascii.hexlify(eth\_hdr[0]))

print(binascii.hexlify(eth\_hdr[1]))

print(binascii.hexlify(eth\_hdr[2]))

# считываем заголовок ip пакета

ipHeader = data[14:34]

ip\_hdr = struct.unpack("!12s4s4s", ipHeader)

# преобразуем адреса в сетевом порядке в строковое представленеи и выводим на экран

print(f"Source IP: {socket.inet\_ntoa(ip\_hdr[1])}")

print(f"Dest IP: {socket.inet\_ntoa(ip\_hdr[2])}")

Далее рассмотрены возможности модуля struct. Сам модуль содержит метод pack и unpack.

Например,

# Упаковать список чисел. Метод pack()

pack\_obj = struct.pack('>4і', LS[0], LS[1], LS[2], LS[3])

# Вывести упакованный объект pack\_obj

print('pack\_obj = ', pack\_obj)

# Распаковать список чисел. Метод unpack().

# Результатом есть кортеж T2

T2 = struct.unpack('>4і', pack\_obj) # T2 = (1, 3, 9, 12)

В языке Python способ упаковки строки определяется на основе первого символа строки формата. Этот символ определяет:

* порядок байт, который формируется с помощью символов @, =, <, >, !. Если не указать этого параметра, то принимается символ @;
* размер в байтах упакованных данных. В этом случае первыми используются цифры, которые обозначают число;
* выравнивание, устанавливаемое системой.

В соответствии с документацией Python в строке формата порядок байт, размер и выравнивание формируются согласно первому символу формата. Возможные первые символы формата отображены в следующей таблице.

|  |  |  |  |
| --- | --- | --- | --- |
| Символ | Порядок байт | Размер | Выравнивание |
| @ | Естественный (зависит от хост-системы) | Естественный | Естественное |
| = | Естественной | Стандартный | Отсутствует |
| < | little-endian | Стандартный | Отсутствует |
| > | big-endian | Стандартный | Отсутствует |
| ! | сетевой (аналог big-endian) | Стандартный | Отсутствует |

Значение порядка байт может быть одним из 4-х:

* естественной порядок (native). Этот порядок может быть или little-endian или big-endian. Данный порядок определяется в зависимости от хост-системы;
* порядок типа little-endian. При таком порядке первым обрабатывается младший байт, а затем уже старший байт;
* порядок типа big-endian. В этом случае первым обрабатывается старший байт, а затем уже младший байт;
* сетевой порядок (network), который по умолчанию равен порядку big-endian.

Размер упакованных данных может быть одним из двух:

* естественный – определяется с помощью инструкции sizeof компилятора языка C;
* стандартный – определяется на основе символа формата в соответствии с нижеследующей таблицей.

Таблица 1 - Определение стандартного размера упакованных данных в зависимости от символа формата

|  |  |  |  |
| --- | --- | --- | --- |
| Формат | Тип в языке C | Тип в языке Python | Стандартный размер |
| x | pad byte | без значения |  |
| c | char | байты длиной 1 | 1 |
| b | signed char | integer | 1 |
| B | unsigned char | integer | 1 |
| ? | \_Bool | bool | 1 |
| h | short | integer | 2 |
| H | unsigned short | integer | 2 |
| i | int | integer | 4 |
| I | unsigned int | integer | 4 |
| l | long | integer | 4 |
| L | unsigned long | integer | 4 |
| q | long long | integer | 8 |
| Q | unsigned long long | integer | 8 |
| n | ssize\_t | integer |  |
| N | size\_t | integer |  |
| e | float (экспоненциальный формат) | float | 2 |
| f | float | float | 4 |
| d | double | float | 8 |
| s | char[] | bytes |  |
| p | char[] | bytes |  |
| P | void\* | integer |  |

##### Примеры форматированных строк для разных типов данных

ii   - два числа типа int

2i   - два числа типа int

10f  - 10 чисел типа float

>i8s - порядок байт big-endian, число типа int, строка из 8 символов

8dif - 8 чисел типа double, 1 число типа int, 1 число типа float

=bi   - естественной порядок, значение типа bool, число типа int

**5.2 Пример использования Scapy**

Scapy — это удобный инструмент для создания пакетов вручную. Утилита написана с использованием языка Python, автором является Philippe Biondi. Возможности утилиты довольны широкие — это и сборка пакетов с последующей отправкой их в сеть, и захват пакетов, и чтение их из сохраненного ранее дампа, и исследование сети. Всё это можно делать как в интерактивном режиме, так и создавая скрипты.

С помощью Scapy можно проводить сканирование, трассировку, исследования, атаки и обнаружение хостов в сети.

Scapy предоставляет среду или даже фреймворк, чем-то похожий на Wireshark, только без красивой графической оболочки.

Утилита разрабатывается под UNIX-подобные операционные системы, но тем не менее, ее можно запустить и в среде Windows.

Можно, например, выяснить поддерживаемые протоколы и поля протоколов.

import scapy.all as sc

# все поддерживаемые протоколы

print(sc.ls())

# поля протокола IP

print(sc.ls(sc.IP))

Cформировать пакеты, вложение указывается через /.

print("Формируем пакет, уровни разделены --")

packet = sc.IP(dst="127.0.0.1")/sc.TCP(dport=80)/"Hello"

# выводим информацию о пакете

print(sc.ls(packet))

# краткая информация о пакете

print(packet.summary())

# более подробная информация

print(packet.show())

# доступ к какому либо слою и полю слоя (например, сегменту TCP и полю flags)

print(f"TCP flags {packet[sc.TCP].flags}")

Перечислим некоторые функции библиотеки.

функция send() – отправляет пакеты, используя сетевой (L3) уровень, никак не обрабатывая ответы. Используется принцип — отправили и забыли;

функция sendp() – отправляет пакеты, используя канальный (L2) уровень, учитываются указанные параметры и заголовки Ethernet кадров. Ответы всё так же не ожидаются и не обрабатываются;

функция sr() – является аналогичной send(), исключение составляет то, что она уже ожидает ответные пакеты;

функция srp() – отправляет и принимает пакеты, уровень L2

функция sr1() – отправляет пакет третьего уровня и забирает только первый ответ, множество ответов не предусматривается;

функция srp1() – аналогично sr1(), только уже канальный уровень.

Пример отправки сформированного пакета.

sc.send(packet)

Получить все пакеты (10 первых) приходящие на интерфейсы (снифер).

def get\_packet(pkt):

print(pkt.summary())

return

sc.sniff(prn = get\_packet, count = 10)

Вывести на экран интерфейсы и получать пакеты с заданного интерфейса с указанным адресом.

print(sc.ifaces)

sc.sniff(prn = get\_packet, filter = "ip host 127.0.0.1", count = 10, iface = "lo", timeout =10)

**5.3 Пример работы с Docker.**

Рассмотрим, как можно запустить ваше приложение в Docker, все ниже перечисленные операции приведены для Ubuntu.

Установим Docker.

Можно зайти по адресу <https://docs.docker.com/desktop/install/ubuntu/> и установить путем скачивания deb файла. Либо инсталлировать используя репозиторий как указано на сайте https://docs.docker.com/engine/install/ubuntu/#set-up-the-repository.

При ручной установке сначала нужно скачать еще файлы в зависимости от версии ubuntu, версию можно посмотреть командой:

lsb\_release -a

Скачиваем, в зависимости от версии linux, например, focal

https://download.docker.com/linux/ubuntu/dists/focal/pool/stable/amd64/.

containerd.io\_<version>\_<arch>.deb

docker-ce\_<version>\_<arch>.deb

docker-ce-cli\_<version>\_<arch>.deb

docker-compose-plugin\_<version>\_<arch>.deb

Устанавливаем

sudo dpkg -i ./containerd.io\_<version>\_<arch>.deb \

./docker-ce\_<version>\_<arch>.deb \

./docker-ce-cli\_<version>\_<arch>.deb \

./docker-compose-plugin\_<version>\_<arch>.deb

Проверяем, что установка прошла успешно

sudo docker run hello-world

В команде указать название файла который вы скачали, либо через sudo dpkg –i названиe.deb.

sudo apt-get update

sudo apt-get install ./docker-desktop-<version>-<arch>.deb

Просмотр информации и команд docker

sudo docker

Просмотр общесистемной информации о docker.

sudo docker info

Просмотр информации о команде docker

sudo docker команда –help

Например

sudo docker exec –help

Поиск доступных образов операционных систем, скрипт пробежится по Docker Hub (репозиторий по умолчанию) и вернет список всех образов с именами, совпадающими со строкой запроса:

sudo docker search ubuntu

В столбце OFFICIAL OK указывает на образ, созданный и поддерживаемый компанией, реализующей проект. После того как вы определили образ, который хотели бы использовать, вы можете загрузить его на свой компьютер с помощью субкоманды pull, например, с учетом установленной у вас операционной системы:

sudo docker pull ubuntu:20.04

Просмотреть имеющиеся образы

sudo docker images

Образы, которые вы используете для запуска контейнеров, можно изменить и использовать для создания новых образов, которые затем могут быть загружены (помещены) на Docker Hub или другие реестры Docker.

Контейнер hello-world служит примером контейнера, который запускается и завершает работу после отправки тестового сообщения. Контейнеры также могут быть интерактивными.

В качестве примера запустим контейнер с нашим образом Ubuntu. Сочетание переключателей -i и -t предоставляет вам доступ к интерактивной командной оболочке внутри контейнера:

sudo docker run -it ubuntu:20.04

Кстати, если запустить sudo docker run -it ubuntu, то установится последняя версия образа latest и добавиться в список образов, можно будет их просмотреть. Выход из контейнера команда exit.

Просмотр активных и неактивных контейнеров:

sudo docker ps

sudo docker ps -a

Запуск и остановку контейнера можно реализовать командами start и stop указав CONTAINER ID или NAME.

Например,

sudo docker start strange\_cerf

sudo docker ps

sudo docker stop 8331dcd04f50

Удаление контейнера docker rm имя\_контейнера.

Можно обновить состояние контейнера установив что-либо там, в результате появится еще один контейнер с установленным уже там дополнительным ПО.

Например в контейнере запустить:

apt update

apt install python

После выхода появится еще один контейнер.

Запустив его получим, контейнер с установленным там Пайтоном (указывайте имя своего контейнера).

sudo docker start -i eager\_heyrovsky

Для того, чтобы скопировать файлы из локальной машины в докер контейнер нужно воспользоваться:

docker cp путь\_на\_локальной\_машине имя\_контейнера:путь\_в\_контейнере

Для обратного процесса выполняем команду:

docker cp имя\_контейнера:путь\_в\_контейнере путь\_на\_локальной\_машине

Например, скопируем все нужные файлы с приложением снифера в наш контейнер с python3.

sudo docker cp /home/user/prog/. eager\_heyrovsky:/home/prog

Запустим контейнер и затем наше приложение.

sudo docker start -i eager\_heyrovsky

Придется установить внутри контейнера

apt install libpcap-dev

apt install pip

apt install python3

pip install scapy

Если есть проблемы с кодировкой, то в запускаемом файле Python можно поставить вначале:

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**5.4 Библиотека SharpPcap**

SharpPcap — библиотека для .NET, которая позволяет перехватывать пакеты. По сути, это обертка над библиотекой Pcap, которая используется во многих популярных продуктах. Например, сниффер Wireshark, IDS Snort.

С SharpPCap также поставляется замечательная библиотека для парсинга пакетов — Packet.Net.

Packet.Net поддерживает следующий протоколы:

\* Ethernet

\* LinuxSLL

\* Ip (IPv4 and IPv6)

\* Tcp

\* Udp

\* ARP

\* ICMPv4 и ICMPv6

\* IGMPv2

\* PPPoE

\* PTP

\* Link Layer Discovery Protocol (LLDP)

\* Wake-On-LAN (WOL)

Работать с библиотекой достаточно просто:

// метод для получения списка устройств

CaptureDeviceList deviceList = CaptureDeviceList.Instance;

// выбираем первое устройство в спсике (для примера)

ICaptureDevice captureDevice = deviceList[0];

// регистрируем событие, которое срабатывает, когда пришел новый пакет

captureDevice.OnPacketArrival += new

PacketArrivalEventHandler(Program\_OnPacketArrival);

// открываем в режиме promiscuous, поддерживается также нормальный режим

captureDevice.Open(DeviceMode.Promiscuous, 1000);

// начинаем захват пакетов

captureDevice.Capture();

Теперь в обработчике события

device\_OnPacketArrival

мы можем работать с пакетом:

static void Program\_OnPacketArrival(object sender, CaptureEventArgs e)

{

// парсинг всего пакета

Packet packet = Packet.ParsePacket(e.Packet.LinkLayerType, e.Packet.Data);

// получение только TCP пакета из всего фрейма

var tcpPacket = TcpPacket.GetEncapsulated(packet);

// получение только IP пакета из всего фрейма

var ipPacket = IpPacket.GetEncapsulated(packet);

if (tcpPacket != null && ipPacket != null)

{

DateTime time = e.Packet.Timeval.Date;

int len = e.Packet.Data.Length;

// IP адрес отправителя

var srcIp = ipPacket.SourceAddress.ToString();

// IP адрес получателя

var dstIp = ipPacket.DestinationAddress.ToString();

// порт отправителя

var srcPort = tcpPacket.SourcePort.ToString();

// порт получателя

var dstPort = tcpPacket.DestinationPort.ToString();

// данные пакета

var data = tcpPacket.PayloadPacket;

}

}

Так же библиотека позволяет создавать пакеты и отправлять, работать с дампами и многое другое. Без проблем работает на mono. У SharpPcap есть конкурент Pcap.net. По описанию возможности совпадают.

**5.5 Драйвер WinPKFilter**

WinPKFilter — NDIS драйвер для перехвата пакетов. Поддерживаются различные операционные системы:x/ME/NT/2000/XP/2003/Vista/2008/Windows 7/2008R2. Плюсом драйвера является то, что он позволяет модифицировать и блокировать пакеты. Для некоммерческих проектов библиотека бесплатна.

Для удобной работы с драйвером предоставляется библиотека. На сайте можно скачать обертки для этой библиотеки для следующих языков — C#, Delphi, VB, MS VC++, C++ Builder.

Работать с WinPkFilter сложнее, чем SharpPcap, нужны хорошие знания в работе с неуправляемым кодов и в маршалинге. Да и размер кода получается намного больше. На официальном сайте можно задать вопрос, на который Вы без проблем получите от автора (кстати русский).

**5.6 Еще один перехватчик с использованием библиотеки scapy на python.**

Далее описан сканер, который будет осуществлять пассивное сканирование сети методом перехвата SSID-идентификаторов.

Весь проект занимает несколько строчек кода на Python, в основу положена библиотека Scapy (www.secdev.org/projects/scapy), предназначенная для манипуляции сетевыми пакетами. К сожалению, под Windows это реализуется на несколько порядков сложнее, поэтому в качестве платформы мы выберем Linux.

Установка scapy

cd /scapy/ & python setup.py install.

Далее можно приступить к написанию сканера. Сканер будет просматривать специальные фреймы, которые содержат уникальный идентификатор сети SSID (Service Set Identifier) и рассылаются точкой доступа. Их также называют Beacon-фреймами. По ним мы и будут определяться найденные сети.

import sys

from scapy import \*

print "Wi-Fi SSID passive sniffer"

interface = sys.argv[1] #задаем название интерфейса в качестве дополнительного консольного аргумента

def sniffBeacon(p):

if p.haslayer(Dot11Beacon):

print p.sprintf("%Dot11.addr2%[%Dot11Elt.info%|%Dot11Beacon.cap%]")

sniff(iface=interface,prn=sniffBeacon)

Вывод программы содержит информацию о перехваченных Beacon-фреймах:

skvoz@box: sniffssid.py eth1

00:12:17:3c:b6:ed['netsquare4'|short-slot+ESS]

00:30:bd:ca:1e:1e['netsquare7'|ESS+privacy]

Также можно поэкспериментировать с выбором haslayer (параметра мониторинга), поменяв его значение Dot11Beacon на: Dot11AssoResp, Dot11ProbeReq, Dot11ATIM, Dot11Auth, Dot11ProbeResp, Dot11Addr2MACField, Dot11Beacon, Dot11ReassoReq, Dot11Addr3MACField, Dot11Deauth, Dot11ReassoResp, Dot11Addr4MACField, Dot11Disas, Dot11WEP, Dot11AddrMACField, Dot11Elt, Dot11AssoReq, Dot11PacketList. В этом случае можно перехватить не только SSID точки доступа, но и всю остальную информацию о сети. К примеру, узнать информацию о физических идентификаторах пользователей и сетевых обращениях. Для этого мы задействуем протокол ARP:

import sys, os

from scapy import \*

interface = raw\_input("enter interface") #пользователь задает интерфейс сети

os.popen("iwconfig interface mode monitor") #перевод карты в режим монитора на заданном интерфейсе

#функция перехвата MAC

def sniffMAC(p):

if p.haslayer(Dot11):

mac = p.sprintf("[%Dot11.addr1%)|(%Dot11.addr2%)|(%Dot11.addr3%)]")

print mac

#функция перехвата IP-адресов и показа ARP сообщений

def sniffarpip(p):

if p.haslayer(IP):

ip = p.sprintf("IP - [%IP.src%)|(%IP.dst%)]")

print ip

elif p.haslayer(ARP):

arp = p.sprintf("ARP - [%ARP.hwsrc%)|(%ARP.psrc%)]-[%ARP.hwdst%)|(%ARP.pdst%)]")

print arp

#уровни, которые мы мониторим

sniff(iface=interface,prn=sniffMAC, prn=sniffarpip)

Вывод

skvoz@puffy: python sniff.py eth1

[ff:ff:ff:ff:ff:ff)|(00:30:bd:ca:1e:1e)|(00:30:bd:ca:1e:1e)]

IP - [192.168.7.41)|(192.168.7.3)]

ARP - [00:0f:a3:1f:b4:ff)|(192.168.7.3)]-[00:00:00:00:00:00)|(192.168.7.41)]

**5.7 Варианты заданий.**

1) Сделать программу для перехвата ICMP и TCP, указать флаги TCP, адрес назначения.

2) Сделать программу для перехвата ICMP и TCP, указать порты TCP, адрес отправителя.

3) Сделать программу для перехвата ICMP и UDP, указать порты UDP, адрес отправителя и получателя.

4) Сделать программу для перехвата ICMP и TCP, указать порты UDP, адрес отправителя и получателя.

5) Сделать программу для перехвата IGMP (формировать самостоятельно) и UDP, указать порты UDP, адрес отправителя и получателя.

6) Сделать программу для перехвата ICMP и TCP, указать адреса.

7) Сделать программу для перехвата ICMP и UDP и TCP указать порты, адрес отправителя и получателя.

8) Сделать программу для перехвата ICMP, сформировать TCP и пакет со временм жизни.

9) Сделать программу для перехвата IP и UDP, указать порты UDP, адрес отправителя и получателя.

10) Сделать программу для перехвата UDP, TCP указать порты UDP, адрес отправителя и получателя.

11) Сделать программу для перехвата IP и TCP, указать порты, адрес отправителя и получателя.

12) Сделать программу для перехвата DNS и UDP, указать порты UDP, адрес отправителя и получателя.

13) Сделать программу для перехвата DNS и ICMP.

14) Сделать программу для перехвата IP и DNS, указать порт, адрес отправителя и получателя.

15) Сделать программу для перехвата TCP и UDP, указать флаги TCP, адрес отправителя и получателя.

16) Сделать программу для перехвата ICMP, IGMP и TCP, указать адрес отправителя и получателя.

17) Сделать программу для перехвата UDP, IP, ICMP, указать порты UDP, адрес отправителя и получателя.

18) Сделать программу для перехвата ARP.

19) Сделать программу для перехвата DNS и ICMP, указать адрес отправителя и получателя, тип запроса ответа ICMP

20) Сделать программу для перехвата ICMP и IP, TCP, указать порты адрес отправителя и получателя, для IP указать NextHeader.