

Утилита для исследования сети и сканер портов Nmap

Абрамов Антон

5 июня 2015 г.

Содержание

1	Цель работы	2
2	Ход работы	2
2.1	Провести поиск активных хостов	2
2.2	Определить открытые порты	2
2.3	Определить версии сервисов	3
2.4	Изучить файлы nmap-services, nmap-os-db, nmap-service-probes	4
2.5	Добавить новую сигнатуру службы в файл nmap-service-probes (для этого создать минимальный tcp server, добиться, чтобы при сканировании nmap указывал для него название и версию).	7
2.6	Сохранить выводы утилиты в формате xml	10
2.7	Исследовать различные этапы и режимы работы nmap с использованием утилиты Wireshark	10
2.8	Просканировать виртуальную машину Metasploitable2 используя nmap_db из состава metasploit-framework	11
2.9	Выбрать пять записей из файла nmap-service-probes и описать их работу. Выбрать один скрипт из состава Nmap и описать его работу	12
3	Выводы	14

1 Цель работы

2 Ход работы

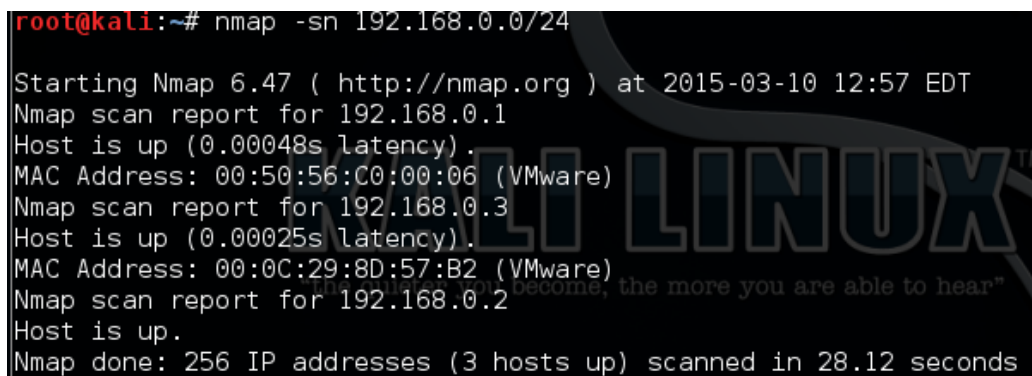
Определить набор и версии сервисов запущенных на компьютере в диапазоне адресов

2.1 Провести поиск активных хостов

Для этого воспользуемся утилитой nmap с ключом sn. Адрес подсети 192.168.0.0/24.

```
nmap -sn 192.168.0.0/24
```

Результат выполнения представлен на рисунке 1.



```
root@kali:~# nmap -sn 192.168.0.0/24
Starting Nmap 6.47 ( http://nmap.org ) at 2015-03-10 12:57 EDT
Nmap scan report for 192.168.0.1
Host is up (0.00048s latency).
MAC Address: 00:50:56:C0:00:06 (VMware)
Nmap scan report for 192.168.0.3
Host is up (0.00025s latency).
MAC Address: 00:0C:29:8D:57:B2 (VMware)
Nmap scan report for 192.168.0.2
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 28.12 seconds
```

Рис. 1: Результат выполнения nmap с ключом -sn

Как видно из результата выполнения у нас есть три активных хоста в сети. Первый хост - это основная ОС. Второй - это ОС Metasploitable2, а третий - это машина, с которой производилось сканирование.

2.2 Определить открытые порты

Определив активные хосты в сети можно их просканировать и определить открытые порты. Сканировать будем ОС Metasploitable2. Это хост с адресом 192.168.0.3. Для сканирования воспользуемся утилитой nmap с ключом p и диапазоном портов от 1 до 65535.

```
nmap -p "*" 192.168.0.3
```

Результат выполнения представлен на рисунке 2.

```
root@kali:~# nmap -p "*" 192.168.0.3

Starting Nmap 6.47 ( http://nmap.org ) at 2015-03-10 13:22 EDT
Nmap scan report for 192.168.0.3
Host is up (0.00030s latency).
Not shown: 4220 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
3632/tcp  open  distccd
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
8180/tcp  open  unknown
MAC Address: 00:0C:29:8D:57:B2 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.67 seconds
```

Рис. 2: Результат выполнения nmap с ключом -p

2.3 Определить версии сервисов

Для определения версии сервисом воспользуемся утилитой nmap с ключом sV.

```
nmap -sV 192.168.0.3
```

Результат выполнения представлен на рисунке 3.

```
Nmap scan report for 192.168.0.3
Host is up (0.0014s latency).
Not shown: 978 closed ports
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          vsftpd 2.3.4
22/tcp    open  ssh          OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp    open  telnet       Linux telnetd
25/tcp    open  smtp         Postfix smtpd
53/tcp    open  domain       ISC BIND 9.4.2
80/tcp    open  http         Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp   open  rpcbind      2 (RPC #100000)
139/tcp   open  netbios-ssn  Samba smbd 3.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn  Samba smbd 3.X (workgroup: WORKGROUP)
512/tcp   open  exec         netkit-rsh rexecd
513/tcp   open  login       
514/tcp   open  shell?
1099/tcp  open  rmiregistry  GNU Classpath grmiregistry
1524/tcp  open  shell        Metasploitable root shell
2049/tcp  open  nfs          2-4 (RPC #100003)
2121/tcp  open  ftp          ProFTPD 1.3.1
3306/tcp  open  mysql        MySQL 5.0.51a-3ubuntu5
5432/tcp  open  postgresql   PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp  open  vnc          VNC (protocol 3.3)
6000/tcp  open  X11          (access denied)
6667/tcp  open  irc          Unreal ircd
8180/tcp  open  unknown
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at http://www.insecure.org/cg
1-bin/servicefp-submit.cgi :
SF-Port514-TCP:V=6.4%I=7%W=D=3/18%Time=54FF243D&P=1686%PC=Linux%P=0%R=(NULL)
SF:33,"%x01getnameinfo:%x20Temporary%x20failure%x20in%x20name%x20resolutio
SF:n\n");
MAC Address: 00:0C:29:8D:57:B2 (VMware)
Service Info: Hosts: metasploitable.localdomain, localhost, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 175.80 seconds
```

Рис. 3: Результат выполнения nmap с ключом -sV

2.4 Изучить файлы nmap-services, nmap-os-db, nmap-service-probes

Откроем файл nmap-services и посмотрим его содержание.

```
nano /usr/share/nmap/nmap-services
```

Этот файл представляет из себя список портов. Внутри находится таблица. Первая колонка - это сервисное название или сокращение. Вторая - число порта и протокол, определенный разделом. Третья колонка - "частота порта мера того, как часто порт был найдет открытым во время сканирования.

Фрагмент файла nmap-services представлен на рисунке 4.

Далее откроем файл nmap-os-db

```
nano /usr/share/nmap/nmap-os-db
```

Файл с данными содержит сотни примеров того, как различные операционные системы отвечают на сканирование. nmap-os-db это база данных с которой сверяются, при определении версии ОС с помощью утилиты nmap с ключа -O.

Фрагмент файла nmap-os-db представлен на рисунке 5.

Теперь откроем файл nmap-service-probes

```
nano /usr/share/nmap/nmap-service-probes
```

nmap-service-probes — это простой текстовый файл состоящий из строк, в котором хранятся тесты и сигнатуры подсистем определений версий. Строки, начинающиеся с символа "решетка" (#) воспринимаются как комментарии и игнорируются обработчиком. Пустые строки также не обрабатываются.

```

yak-chat      258/udp 0.000494      # yak winsock personal chat
esro-gen      259/tcp 0.000201      # efficient short remote operations
firewall1-rdp 259/udp 0.000840      # Firewall 1 proprietary RDP protocol
openport      260/tcp 0.000025
openport      260/udp 0.000362
nsiiops 261/tcp 0.000025      # iiop name service over tls/ssl
nsiiops 261/udp 0.000659      # iiop name service over tls/ssl
arcisdms      262/tcp 0.000038
arcisdms      262/udp 0.000577
hdap          263/udp 0.000544
bgmp          264/tcp 0.001029
fwl-or-bgmp    264/udp 0.000461      # FWL secureremote alternate
maybe-fwl    265/tcp 0.000013
td-service     267/tcp 0.000013      # Tobit David Service Layer
td-replica     268/tcp 0.000050      # Tobit David Replica
unknown 270/tcp 0.000013
unknown 271/tcp 0.000013

```

Рис. 4: Фрагмент файла nmap-services

```

# iPod Touch 16Gb. With 1.1.3 upgrade,BSD Subsystem, Openssh, and some other utils. Darwin 9.0.0d1 Darwin Kernel Version 9.0.0d1: Wed Oct 10 00:07:56
Fingerprint Apple iPod touch audio player (iPhone OS 1.1.3, Darwin 9.0.0d1)
Class Apple | iPhone OS | 1.X | media device
CPE cpe:/o:apple:iphone_os:1:0:auto
SEQ(SP=0-5%CD=1-6%ISR=15-16%TS=1)
OPS(O1=M3FD8NW0NNT11SLL%02=M3FD8NW0NNT11SLL%03=M3FD8NW0NNT11%04=M3FD8NW0NNT11SLL%05=M3FD8NW0NNT11SLL%06=M3FD8NW0NNT11SLL)
WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)
ECN(R=Y%DF=N|Y%T=3B-45%TG=40%W=0|FFFF%0=|M3FD8NW0SLL%CC=N%Q=)
T1(R=Y%DF=Y%T=3B-45%TG=40%W=0%A=S+SF=AR%RD=0%Q=)
T2(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=Z%A=SF=AR%RD=0%Q=)
T3(R=N)
T4(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=AR%Z%F=AR%RD=0%Q=)
T5(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=Z%A-S+SF=AR%RD=0%Q=)
T6(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=AR%Z%F=AR%RD=0%Q=)
T7(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=Z%A-S+SF=AR%RD=0%Q=)
U1(DF=N%T=3B-45%TG=40%IPL=38%UN=0%RIPL=0%RID=0%RIPCK=Z%RUCK=0%RUD=G)
IE(DFI=S%T=3B-45%TG=40%CD=S)

```

Рис. 5: Фрагмент файла nmap-os-db

Синтаксис:

- **Probe** <protocol> <probenam> <probesendstring>

Директива probe (тест) указывает nmap, какие данные отправлять в процессе определения служб. Пример использования директивы Probe представлен на рисунке 6.

```

#####NEXT PROBE#####
# Quake 3 and other games
# http://svn.icculus.org/twilight/trunk/dpmaster/doc/techinfo.txt?view=markup
# Protocol 68 is a specific revision of Quake 3, but the server should respond
# with an empty server list even if it doesn't know that game.
Probe UDP Quake3_master_getservers q|\xff\xff\xff\xffgetservers 68 empty full|
rarity 9
ports 27950,30710
match quake3-master m|^\\xff\\xff\\xff\\xffgetserversResponse|

```

Рис. 6: Пример использования директивы Probe

- **match** <service> <pattern> <productname> <version> <device>
<h?????> <info> <OS>

Директива **match** указывает nmap на то, как точно определить службу, используя полученный ответ на запрос, отправленный предыдущей директивой **probe**. Эта директива используется в случае, когда полученный ответ полностью совпадает с шаблоном. При этом тестирование порта считается законченным, а при помощи дополнительных спецификаторов nmap строит отчет о названии приложения, номере версии и дополнительной информации, полученной в ходе проверки. Пример использования директивы **match** представлен на рисунке 7.

```
match kumo-server m|^x94\x01\xcd\xef\xd1\xc0\xda\0.([\s]+)|s p/Kumofs/ v/$1/
match kumo-manager m|^x94\x01\xcd\xef\xd1\x05\xc0$| p/Kumofs/
```

Рис. 7: Пример использования директивы **match**

- **softmatch** <service> <pattern> <productname> <version> <device>
<h?????> <info> <OS>

Директива **softmatch** имеет аналогичный формат директиве **match**. Основное отличие заключается в том, что после совпадения принятого ответа с одним из шаблонов **softmatch**, тестирование будет продолжено с использованием только тех тестов, которые относятся к определенной шаблоном службе. Тестирование порта будет идти до тех пор, пока не будет найдено строгое соответствие (**match**) или не закончатся все тесты для данной службы. Пример использования директивы **softmatch** представлен на рисунке 8.

```
softmatch ibm-mqseries m|^TSH\x20\0\0\0| p/IBM WebSphere MQ/ cpe:/a:ibm:websphere_mq/
```

Рис. 8: Пример использования директивы **softmatch**

- **ports** <portlist>

Директива **ports** группирует порты, которые обычно закрепляются за идентифицируемой данным тестом службой. Синтаксис представляет собой упрощенный формат опции **-p**. Пример использования директивы **ports** представлен на рисунке 9.

- **sslports** <sslportlist>

```
ports 5000,5001,5002,10001
```

Рис. 9: Пример использования директивы ports

Директива sslports аналогична директиве ports, только эта директива указывает порты, обычно используемые совместно с SSL. Пример использования директивы sslports представлен на рисунке 10.

```
sslports 55553
```

Рис. 10: Пример использования директивы sslports

- **totalwaitms** <milliseconds>

Директива totalwaitms редко используемая, т.к. указывает сколько времени (в миллисекундах) необходимо ждать ответ, прежде чем прекратить тест службы. Пример использования директивы totalwaitms представлен на рисунке 11.

```
totalwaitms 11000
```

Рис. 11: Пример использования директивы totalwaitms

2.5 Добавить новую сигнатуру службы в файл nmap-service-probes (для этого создать минимальный tcp server, добиться, чтобы при сканировании nmap указывал для него название и версию).

Создадим минимальный tcp сервер. Сервер написан на языке C# и прослушивает порт 9595. Запустим его на основной операционной системе, сетевой адрес которой 192.168.0.1. Сервер будет ожидать подключения и отвечать на "нуль-тест" определенным сообщением. В файле nmap-service-probes добавим сигнатуру нашей подсистемы. Т.к. "нуль-тест" пройдет удачно и произойдет полное совпадение с сигнатурой, что означает завершение тестирования, не требуется создавать ответы на другие тесты.

Код сервера:

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;

namespace ExampleTcpListener_Console
{
    class ExampleTcpListener
    {
        static void Main(string[] args)
        {
            TcpListener server = null;
            try
            {
                // Определим нужное максимальное количество потоков
                // Пусть будет по 4 на каждый процессор
                int MaxThreadsCount = Environment.ProcessorCount * 4;
                Console.WriteLine(MaxThreadsCount.ToString());
                // Установим максимальное количество рабочих потоков
                ThreadPool.SetMaxThreads(MaxThreadsCount, MaxThreadsCount);
                // Установим минимальное количество рабочих потоков
                ThreadPool.SetMinThreads(2, 2);

                // Устанавливаем порт для TcpListener = 9595.
                Int32 port = 9595;
                IPAddress localAddr = IPAddress.Parse("192.168.0.1");
                int counter = 0;
                server = new TcpListener(localAddr, port);

                // Запускаем TcpListener и начинаем слушать клиентов.
                server.Start();

                // Принимаем клиентов в бесконечном цикле.
                while (true)
                {
                    Console.WriteLine("\nWaiting for a connection... ");
                    // При появлении клиента добавляем в очередь потоков его обр
                    ThreadPool.QueueUserWorkItem(ObrabotkaZaprosa, server.Accept

```



```

        // Выводим информацию о подключении.
        counter++;
        Console.WriteLine("\nConnection №" + counter.ToString() + "!");
    }
}
catch (SocketException e)
{
    //В случае ошибки, выводим что это за ошибка.
    Console.WriteLine("SocketException: {0}", e);
}
finally
{
    // Останавливаем TcpListener.
    server.Stop();
}
Console.WriteLine("\nHit enter to continue...");
Console.Read();
}

static void ObrabotkaZaprosa(object client_obj)
{
    // Буфер для принимаемых данных.
    Byte[] bytes = new Byte[256];
    String data = null;

    //Можно раскомментировать Thread.Sleep(1000);
    //Запустить несколько клиентов
    //и наглядно увидеть как они обрабатываются в очереди.
    //Thread.Sleep(1000);

    TcpClient client = client_obj as TcpClient;

    data = null;

    // Получаем информацию от клиента
    NetworkStream stream = client.GetStream();

    int i;
    /*
    data = "zzzzzz";
    byte[] msg = System.Text.Encoding.ASCII.GetBytes(data);

```

```

        stream.Write(msg, 0, msg.Length);

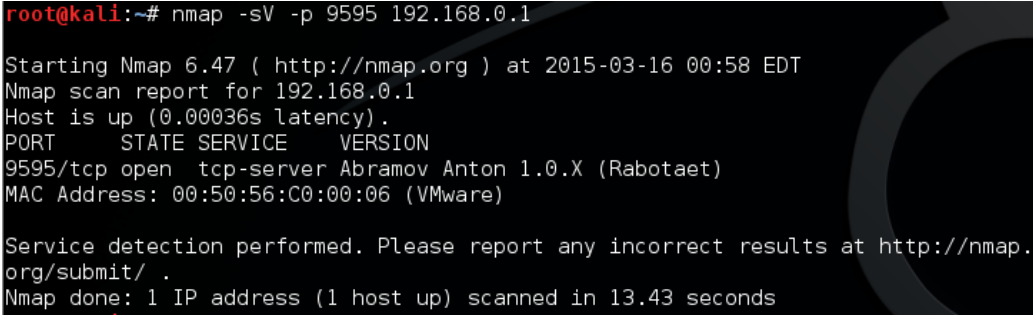
        // Закрываем соединение.
        client.Close();
    }
}
}

```

Текст, дописанный в файл `nmap-service-probes`:
`match tcp-server m|^zzzzzz| v/1.0.X/ p/Abramov Anton/ i/Rabotaet/`
 Запустим сервер, на основной ОС, а на kali Linux воспользуемся ути-
 литой `nmap`:

```
nmap -sV -p 9595 192.168.0.1
```

Результат выполнения представлен на рисунке 12:



```

root@kali:~# nmap -sV -p 9595 192.168.0.1

Starting Nmap 6.47 ( http://nmap.org ) at 2015-03-16 00:58 EDT
Nmap scan report for 192.168.0.1
Host is up (0.00036s latency).
PORT      STATE SERVICE      VERSION
9595/tcp  open  tcp-server  Abramov Anton 1.0.X (Rabotaet)
MAC Address: 00:50:56:C0:00:06 (VMware)

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.43 seconds

```

Рис. 12: Демонстрация работы новой сигнатуры

2.6 Сохранить выводы утилиты в формате xml

Для этого воспользуемся утилитой `nmap` с ключом `-oX`.

```
nmap -sn -oX output.xml 192.168.0.3
```

Вывод произведен в файл `output.xml`. Результат выполнения утилиты продемонстрирован на рисунке 13.

2.7 Исследовать различные этапы и режимы работы nmap с использованием утилиты Wireshark

Wireshark работает с подавляющим большинством известных протоколов, имеет понятный и логичный графический интерфейс на основе GTK+ и мощнейшую систему фильтров.

```

<?xml version="1.0"?>
<!DOCTYPE nmaprun>
<?xml-stylesheet href="file:///usr/bin/./share/nmap/nmap.xsl" type="text/xsl"?>
<!-- Nmap 6.47 scan initiated Tue Mar 10 19:24:28 2015 as: nmap -sn -oX output.xml 192.168.0.3 -->
<nmaprun scanner="nmap" args="nmap -sn -oX output.xml 192.168.0.3" start="1426029868" startstr="Tue Mar 10 19:24:28 2015" version="6.47" xmloutputversion="1.04">
  <verbose level="0"/>
  <debugging level="0"/>
  <host><status state="up" reason="arp-response" reason_ttl="0"/>
  <address addr="192.168.0.3" addrtype="ipv4"/>
  <address addr="00:0C:29:8D:57:B2" addrtype="mac" vendor="VMware"/>
  <hostnames>
  </hostnames>
  <times srtt="499" rttvar="5000" to="100000"/>
</host>
<runstats><finished time="1426029881" timestr="Tue Mar 10 19:24:41 2015" elapsed="13.08" summary="Nmap done at Tue Mar 10 19:24:41 2015; 1 IP address (1 host up) scanned in 13.08 seconds" exit="success"/><hosts up="1" down="0" total="1"/>
</runstats>
</nmaprun>

```

Рис. 13: Результат выполнения утилиты nmap с ключом -oX

Запустим утилиту Wireshark, выберем интерфейс eth0 и нажмем Start. Будет произведено сканирование сети и вывод передаваемых по сети пакетов. Сканирование сети производится путем сканирования каждого из портов. На порт отправляется запрос, если порт открыт и прослушивается, то мы сможем получить ответ на свой запрос. Однако ответ можно получить не всегда, на порту может стоять защита, запрещающая отвечать на пустые запросы. В таком случае мы ничего не получим, что означает функционирование порта, но с определенной защитой. В таком случае могут посылаются запросы с определенным сообщением. В обычных случаях, если порт открыт, он будет отправлять ответ идентифицирующий приложение или службу, работающие на порту. Wireshark по-умолчанию выводит все пакеты, которые проходят через интерфейс eth0. Все пакеты просматривать неудобно, поэтому мы будем пользоваться фильтрами. Поставим фильтр по протоколу tcp. Пропингуем с ОС Metasploitable2 основную ОС и проверим, будет ли утилита Wireshark видеть подобную активность сети? Как видно из рисунка 14, утилита отображает информацию об активности в сети.

2.8 Просканировать виртуальную машину Metasploitable2 используя nmap_db из состава metasploit-framework

Запустив **metasploit**, мы можем воспользоваться командой **db_nmap**.

No.	Time	Source	Destination	Protocol	Length	Info
562	336.4733900	192.168.0.3	192.168.0.1	ICMP	98	Echo (ping) request id=0x8015, seq=21/5376, ttl=64
563	336.4734880	192.168.0.1	192.168.0.3	ICMP	98	Echo (ping) reply id=0x8015, seq=21/5376, ttl=128 (request in 562)
564	337.3079630	192.168.0.1	192.168.0.255	BROWSER	235	Browser Election Request
565	337.4825660	192.168.0.3	192.168.0.1	ICMP	98	Echo (ping) request id=0x8015, seq=22/5632, ttl=64
566	337.4827780	192.168.0.1	192.168.0.3	ICMP	98	Echo (ping) reply id=0x8015, seq=22/5632, ttl=128 (request in 565)
567	338.3076500	192.168.0.1	192.168.0.255	NBNS	110	Registration NB 322<id>
568	338.4926480	192.168.0.3	192.168.0.1	ICMP	98	Echo (ping) request id=0x8015, seq=23/5888, ttl=64
569	338.4928110	192.168.0.1	192.168.0.3	ICMP	98	Echo (ping) reply id=0x8015, seq=23/5888, ttl=128 (request in 568)
570	339.0576840	192.168.0.1	192.168.0.255	NBNS	110	Registration NB 322<id>
571	339.5018650	192.168.0.3	192.168.0.1	ICMP	98	Echo (ping) request id=0x8015, seq=24/6144, ttl=64
572	339.5020890	192.168.0.1	192.168.0.3	ICMP	98	Echo (ping) reply id=0x8015, seq=24/6144, ttl=128 (request in 571)
573	339.8077790	192.168.0.1	192.168.0.255	NBNS	110	Registration NB 322<id>
574	340.5128590	192.168.0.3	192.168.0.1	ICMP	98	Echo (ping) request id=0x8015, seq=25/6400, ttl=64 (reply in 575)
575	340.5128820	192.168.0.1	192.168.0.3	ICMP	98	Echo (ping) reply id=0x8015, seq=25/6400, ttl=128 (request in 574)
576	340.5577500	192.168.0.1	192.168.0.255	NBNS	110	Registration NB 322<id>

Frame 3: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits) on interface 0
 Ethernet II, Src: Vmware_Bd:57:b2 (00:0c:29:8d:57:b2), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 Internet Protocol Version 4, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
 User Datagram Protocol, Src Port: bootps (68), Dst Port: bootps (67)
 Bootstrap Protocol

Рис. 14: Работа утилиты Wireshark

2.9 Выбрать пять записей из файла nmap-service-probes и описать их работу. Выбрать один скрипт из состава Nmap и описать его работу

1. Запись первая

На рисунке 15 мы видим первую запись. Тут используется директива **probe**, отправляющая "нуль тест"(пустое сообщение) по протоколу TCP. Далее используется директива **totalwaitms** с величиной в 6 секунд, что означает, что "нуль тест" будет ожидать ответа в течении шести секунд. Если ответа не поступит, то будут применены другие тесты.

```
# This is the NULL probe that just compares any banners given to us
#####NEXT PROBE#####
Probe TCP NULL q||
# Wait for at least 6 seconds for data. It used to be 5, but some
# smtp services have lately been instituting an artificial pause (see
# FEATURE('greet_pause') in Sendmail, for example)
totalwaitms 6000
```

Рис. 15: Запись первая, "нуль тест"

2. Запись вторая

На рисунке 16 видим вторую запись. Тут используется директива **match**, определяющая службу, по полученному ответу на запрос. Служба называется **activemq**, ее ответ имеет следующий вид: `\0\0\0.\x01ActiveMQ\0\0\0`. А название производителя или имя службы **Apache ActiveMQ**.

```
match activemq m|^0\0\0.\x01ActiveMQ\0\0\0|s p/Apache ActiveMQ/
```

Рис. 16: Запись вторая. Apache ActiveMQ

3. Запись третья

На рисунке 17 видим третью запись. Тут используется директива **match**. Служба с именем **dnsix** отвечает следующим образом: **DNSIX\$**.

```
match dnsix m|^DNSIX$|
```

Рис. 17: Запись третья, dnsix

4. Запись четвертая

На рисунке 18 видим четвертую запись. Тут используется директива **softmatch**, определяющая службу, по полученному ответу на запрос, но в отличии от **match**, в этом случае тестировании будет продолжено. Служба называется **gkrellm**, ее ответ имеет следующий вид: **<error>\nConnection not allowed from .*\n**. А название производителя или имя службы **GKrellM System Monitor**

```
softmatch gkrellm m|^<error>\nConnection not allowed from .*\n| p/GKrellM System Monitor/
```

Рис. 18: Запись четвертая, GKrellM System Monitor

5. Запись пятая

На рисунке 19 видим пятую запись. Тут использована директива **match**, определяющая службу, по полученному ответу на запрос. Служба называется **starutil**, ее ответ имеет следующий вид: **star-v3 stility server\n\n0**. Название производителя или имя службы **StarUTIL route config**, версия службы **3**, название устройства **router**

```
match starutil m|^star-v3 utility server\n\n0| p/StarUTIL route config/ v/3/ d/router/
```

Рис. 19: Запись пятая, StarUTIL route config

6. Скрипт x11-access

Назначение:

Проверяет, есть ли у вас разрешение подключаться к X-серверу. Если x-сервер прослушивает TCP-порт 6000+n (где n число дисплеев), это можно проверить, если вы сможете подключиться к удаленному дисплею, отправив первоначальный запрос на подключение x11.

При этом будет получен ответ (0x00 или 0x01).

Пример использования:

```
nmap -sV -sC <target>
```

Вывод скрипта :

Host script results:

```
|_ x11-access: X server access is granted
```

3 Выводы

В ходе выполнения данной лабораторной работы мы познакомились с утилитой nmap и Wireshark. Рассмотрели возможности этих утилит, а также файлы, с которыми эти утилиты работают. В частности, были рассмотрены файлы nmap-services, nmap-os-db, nmap-service-probes. В качестве сканируемой операционной системы была выбрана Metasploitable2.