Peter the Great St.Petersburg Polytechnic University

Department of Computer Systems & Software Engineering

Laboratory report №2

Discipline: «Information Security»

Theme: «Network Mapper»

Made by student:

Boyarkin N.S.
Group: 13541/3

Lecturer:

Bogach N.V.

Saint-Petersburg
2017 y.

Contents

1 Laboratory work №2 **2**
1.1 Work purpose . . . 2
1.2 Task . . . 2
1.3 Work Progress . . . 3
1.3.1 Introduction . . . 3
1.3.2 Creating connection . . . 3
1.3.3 List targets to scan . . . 4
1.3.4 Probe open ports to determine service/version info . . . 4
1.3.5 Study nmap-services, nmap-os-db, nmap-service-probes . . . 6
1.3.6 Add new service to nmap-service-probes . . . 8
1.3.7 Study nmap stages and modes using Wireshark . . . 13
1.3.8 Output to xml-format file . . . 14
1.3.9 Perform VM Metasploitable2 scanning using db_nmap from metasploitframework . . . 14
1.3.10 Get some records from nmap-service-probes and describe them . . . 15
1.3.11 Choose one Nmap Script and describe it . . . 16
1.4 Conclusion . . . 17

Laboratory work №2

1.1 Work purpose

Study nmap utility with help of Kali Linux and Metasploitable2 VM.

1.2 Task

1. List targets to scan.
2. Probe open ports to determine service/version info.
3. Study nmap-services, nmap-os-db, nmap-service-probes.
4. Add new service to nmap-service-probes (create a minimal tcp server, get its name and version by nmap).
5. Study nmap stages and modes using Wireshark.
6. Output to xml-format file.
7. Perform VM Metasploitable2 scanning using db_nmap from metasploitframework.
8. Get some records from nmap-service-probes and describe them. Choose one Nmap Script and describe it.

1.3 Work Progress

1.3.1 Introduction

Nmap uses raw IP packets in novel ways to determine what hosts are availableon the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. It was designed to rapidly scan large networks, but works fine against single hosts. Nmap runs on all major computer operating systems, and official binary packages are available for Linux, Windows, and Mac OS X.

1.3.2 Creating connection

We used three operating systems at this laboratory work:

1. **Kali Linux** – VM, IP: 192.168.56.102
2. **Metasploitable 2** – VM, IP: 192.168.56.101
3. **Windows 10** – Real, IP: 192.168.56.1

The following code obtains the network configuration for Kali Linux:

```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.102  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fe81:b1df  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:81:b1:df  txqueuelen 1000  (Ethernet)
        RX packets 7  bytes 1787 (1.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 2312 (2.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 24  bytes 1440 (1.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 24  bytes 1440 (1.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Network configuration for Metasploitable 2:

```
msfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:7e:e4:cc
          inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe7e:e4cc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:276 errors:0 dropped:0 overruns:0 frame:0
          TX packets:442 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:27731 (27.0 KB)  TX bytes:89510 (87.4 KB)
          Base address:0xd010 Memory:f0000000-f0020000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:118 errors:0 dropped:0 overruns:0 frame:0
          TX packets:118 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:31841 (31.0 KB)  TX bytes:31841 (31.0 KB)
```

Рис. 1.1: Metasploitable 2 network configuration

Let's try to check connection between Kali and Metasploitable 2 by the browser:

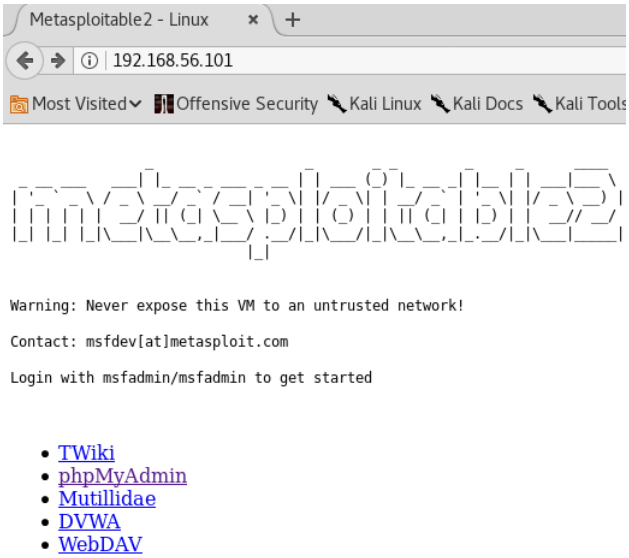

Рис. 1.2: Connection from Kali to Metasploitable 2 established

Ping also going well:

```
msfadmin@metasploitable:~$ ping 192.168.56.102
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data.
64 bytes from 192.168.56.102: icmp_seq=1 ttl=64 time=0.166 ms
64 bytes from 192.168.56.102: icmp_seq=2 ttl=64 time=0.263 ms
64 bytes from 192.168.56.102: icmp_seq=3 ttl=64 time=0.238 ms
```

Рис. 1.3: Connection from Metasploitable 2 to Kali established

1.3.3 List targets to scan

Let's start scaning Metasploitable 2 OS by the **nmap** utility. Option **-n** initiates fast scan (without port scanning). The main argument of the nmap utility is IP address (or range) of the remote host.

```
root@kali:~# nmap -sn 192.168.56.101

Starting Nmap 7.60 ( https://nmap.org ) at 2017-11-05 14:24 EST
Nmap scan report for 192.168.56.101
Host is up (0.00015s latency).
MAC Address: 08:00:27:7E:E4:CC (Oracle VirtualBox virtual NIC)
Nmap done: 1 IP address (1 host up) scanned in 13.28 seconds
```

1.3.4 Probe open ports to determine service/version info

Option **-top-ports** searches for the most used ports of the remote machine. Let's compare the result of nmap utility for the Metasploitable 2 OS and Windows 10 OS.

```
root@kali:~# nmap -top-ports 5 192.168.56.101

Starting Nmap 7.60 ( https://nmap.org ) at 2017-11-05 14:29 EST
Nmap scan report for 192.168.56.101
Host is up (0.00014s latency).

PORT    STATE  SERVICE
21/tcp  open   ftp
22/tcp  open   ssh
23/tcp  open   telnet
80/tcp  open   http
443/tcp closed https
MAC Address: 08:00:27:7E:E4:CC (Oracle VirtualBox virtual NIC)
```

```
14
15 Nmap done: 1 IP address (1 host up) scanned in 13.55 seconds
16
17
18 root@kali:~# nmap -top-ports 5 192.168.56.1
19
20 Starting Nmap 7.60 ( https://nmap.org ) at 2017-11-05 14:33 EST
21 Nmap scan report for 192.168.56.1
22 Host is up (0.0011s latency).
23
24 PORT    STATE    SERVICE
25 21/tcp  filtered ftp
26 22/tcp  open     ssh
27 23/tcp  filtered telnet
28 80/tcp  filtered http
29 443/tcp filtered https
30 MAC Address: 0A:00:27:00:00:02 (Unknown)
31
32 Nmap done: 1 IP address (1 host up) scanned in 14.56 seconds
```

Metasploitable 2 has many vulnerabilities, including open ports without filtering. At the same time Windows has a built-in firewall, that filters some nmap packets.

Option **-V** used to display versions of the protocols:

```
1  root@kali:~# nmap -sV -top-ports 5 192.168.56.101
2
3  Starting Nmap 7.60 ( https://nmap.org ) at 2017-11-05 14:56 EST
4  Nmap scan report for 192.168.56.101
5  Host is up (0.00014s latency).
6
7  PORT    STATE  SERVICE VERSION
8  21/tcp  open   ftp     vsftpd 2.3.4
9  22/tcp  open   ssh     OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
10 23/tcp  open   telnet  Linux telnetd
11 80/tcp  open   http    Apache httpd 2.2.8 ((Ubuntu) DAV/2)
12 443/tcp closed https
13 MAC Address: 08:00:27:7E:E4:CC (Oracle VirtualBox virtual NIC)
14 Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
15
16 Service detection performed. Please report any incorrect results at https://nmap.org/
       submit/ .
17 Nmap done: 1 IP address (1 host up) scanned in 19.84 seconds
18
19
20 root@kali:~# nmap -sV -top-ports 5 192.168.56.1
21
22 Starting Nmap 7.60 ( https://nmap.org ) at 2017-11-05 14:57 EST
23 Nmap scan report for 192.168.56.1
24 Host is up (0.00015s latency).
25
26 PORT    STATE    SERVICE VERSION
27 21/tcp  filtered ftp
28 22/tcp  open     ssh     Microsoft Windows IoT sshd 1.100 (protocol 2.0)
29 23/tcp  filtered telnet
30 80/tcp  filtered http
31 443/tcp filtered https
32 MAC Address: 0A:00:27:00:00:02 (Unknown)
33 Service Info: OS: Windows 10 IoT Core; CPE: cpe:/o:microsoft:windows_10:::iot_core
34
35 Service detection performed. Please report any incorrect results at https://nmap.org/
       submit/ .
36 Nmap done: 1 IP address (1 host up) scanned in 14.91 seconds
```

This experiment shows that remote attackers can get a lot of information about the system, so to ensure security, you must use a firewall.

1.3.5 Study nmap-services, nmap-os-db, nmap-service-probes

The utility files for nmap can be found in the directory /**usr**/**share**/**nmap**.

The **nmap-services** file is a registry of port names to their corresponding number and protocol. Each entry has a number representing how likely that port is to be found open. Most lines have a comment as well. Nmap ignores the comments, but users sometimes grep for them in the file when Nmap reports an open service of a type that the user does not recognize.

```
root@kali:~# cat /usr/share/nmap/nmap-services
tcpmux  1/tcp 0.001995  # TCP Port Service Multiplexer [rfc-1078] | TCP Port Service
    Multiplexer
tcpmux  1/udp 0.001236  # TCP Port Service Multiplexer
compressnet 2/tcp 0.000013  # Management Utility
compressnet 2/udp 0.001845  # Management Utility
compressnet 3/tcp 0.001242  # Compression Process
compressnet 3/udp 0.001532  # Compression Process
unknown 4/tcp 0.000477
rje 5/tcp 0.000000  # Remote Job Entry
rje 5/udp 0.000593  # Remote Job Entry
unknown 6/tcp 0.000502
echo  7/sctp  0.000000
echo  7/tcp 0.004855
echo  7/udp 0.024679
unknown 8/tcp 0.000013
discard 9/sctp  0.000000  # sink null
discard 9/tcp 0.003764  # sink null
discard 9/udp 0.015733  # sink null
unknown 10/tcp  0.000063
systat  11/tcp  0.000075  # Active Users
systat  11/udp  0.000577  # Active Users
unknown 12/tcp  0.000063
daytime 13/tcp  0.003927
daytime 13/udp  0.004827
unknown 14/tcp  0.000038
netstat 15/tcp  0.000038
unknown 16/tcp  0.000050
qotd  17/tcp  0.002346  # Quote of the Day
qotd  17/udp  0.009209  # Quote of the Day
< ... >
```

The **nmap-os-db data** file contains hundreds of examples of how different operating systems respond to Nmap's specialized OS detection probes. It is divided into blocks known as fingerprints, with each fingerprint containing an operating system's name, its general classification, and response data.

```
root@kali:~# cat /usr/share/nmap/nmap-os-db
MatchPoints
SEQ(SP=25%GCD=75%ISR=25%TI=100%CI=50%II=100%SS=80%TS=100)
OPS(O1=20%O2=20%O3=20%O4=20%O5=20%O6=20)
WIN(W1=15%W2=15%W3=15%W4=15%W5=15%W6=15)
ECN(R=100%DF=20%T=15%TG=15%W=15%O=15%CC=100%Q=20)
T1(R=100%DF=20%T=15%TG=15%S=20%A=20%F=30%RD=20%Q=20)
T2(R=80%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
T3(R=80%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
T4(R=100%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
T5(R=100%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
T6(R=100%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
T7(R=80%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
U1(R=50%DF=20%T=15%TG=15%IPL=100%UN=100%RIPL=100%RID=100%RIPCK=100%RUCK=100%RUD=100)
IE(R=50%DFI=40%T=15%TG=15%CD=100)

# 2N VOIP doorbell
Fingerprint 2N Helios IP VoIP doorbell
Class 2N | embedded || specialized
CPE cpe:/h:2n:helios
SEQ(SP=0-5%GCD=51E80C|A3D018|F5B824|147A030|199883C%ISR=C8-D2%TI=I|RD%CI=I%II=RI%SS=S%TS=
    U)
OPS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)
```

```
WIN(W1=8000%W2=8000%W3=8000%W4=8000%W5=8000%W6=8000)
ECN(R=Y%DF=N%T=FA-104%TG=FF%W=8000%O=M5B4%CC=N%Q=)
T1(R=Y%DF=N%T=FA-104%TG=FF%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=N)
T3(R=Y%DF=N%T=FA-104%TG=FF%W=8000%S=O%A=S+%F=AS%O=M5B4%RD=0%Q=)
T4(R=Y%DF=N%T=FA-104%TG=FF%W=8000%S=A+%A=S%F=AR%O=%RD=0%Q=)
T5(R=Y%DF=N%T=FA-104%TG=FF%W=8000%S=A%A=S+%F=AR%O=%RD=0%Q=)
T6(R=Y%DF=N%T=FA-104%TG=FF%W=8000%S=A%A=S%F=AR%O=%RD=0%Q=)
T7(R=Y%DF=N%T=FA-104%TG=FF%W=8000%S=A%A=S+%F=AR%O=%RD=0%Q=)
U1(DF=N%T=FA-104%TG=FF%IPL=38%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)
IE(DFI=S%T=FA-104%TG=FF%CD=S)

# BT2700HGV DSL Router version 5.29.107.19
Fingerprint 2Wire BT2700HG-V ADSL modem
Class 2Wire | embedded || broadband router
CPE cpe:/h:2wire:bt2700hg-v
SEQ(SP=6A-BE%GCD=1-6%ISR=96-A0%TI=I%CI=I%II=I%SS=S%TS=A)
OPS(O1=M5B4NNSW0NNNT11%O2=M578NNSW0NNNT11%O3=M280W0NNNT11%O4=M218NNSW0NNNT11%O5=
    M218NNSW0NNNT11%O6=M109NNSNNT11)
WIN(W1=8000%W2=8000%W3=8000%W4=8000%W5=8000%W6=8000)
ECN(R=Y%DF=Y%T=FA-104%TG=FF%W=8000%O=M5B4NNSW0N%CC=N%Q=)
T1(R=Y%DF=Y%T=FA-104%TG=FF%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=N)
T3(R=N)
T4(R=Y%DF=Y%T=FA-104%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=E44A4E43%Q=)
T5(R=Y%DF=Y%T=FA-104%TG=FF%W=0%S=Z%A=S+%F=AR%O=%RD=1F59B3D4%Q=)
T6(R=Y%DF=Y%T=FA-104%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=1F59B3D4%Q=)
T7(R=N)
U1(DF=Y%T=FA-104%TG=FF%IPL=70%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)
IE(DFI=Y%T=FA-104%TG=FF%CD=S)

< ... >
```

While the version of nmap-services distributed with Nmap is sufficient for most users, understanding the file format allows advanced Nmap hackers to add their own services to the detection engine. Like many Unix files, **nmap-service-probes** is line-oriented. Lines starting with a hash (#) are treated as comments and ignored by the parser. Blank lines are ignored as well. Other lines must contain one of the directives described below.

```
root@kali:~# cat /usr/share/nmap/nmap-service-probes
tcpwrappedms 3000

match 1c-server m|^S\xf5\xc6\x1a{| p/1C:Enterprise business management server/

match 4d-server m|^\0\0\0H\0\0\0\x02
    .[^\0]*\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
    $|s p/4th Dimension database server/ cpe:/a:4d_sas:4d/

match aastra-pbx m|^BUSY$| p|Aastra/Mitel 400-series PBX service port|
match acap m|^\* ACAP \(IMPLEMENTATION \"CommuniGate Pro ACAP (\d[-.\w]+)\"\) | p/
    CommuniGate Pro ACAP server/ v/$1/ i/for mail client preference sharing/ cpe:/a:
    stalker:communigate_pro:$1/
match acarsd m|^g\0\0\0\x1b\0\0\0\0\0\0\0acarsd\t([\w._-]+)\tAPI-([\w._-]+)\)\0\0\0\x06\
    x05\0\0\0\0\0\0<\?xml | p/acarsd/ v/$1/ i/API $2/ cpe:/a:acarsd:acarsd:$1/
match acmp m|^ACMP Server Version ([\w._-]+)\r\n| p/Aagon ACMP Inventory/ v/$1/

match apachemq m|^\0\0..\x01ActiveMQ\0\0\0.\x01\0\0.*\x0cProviderName\t\0\x08ActiveMQ.*\
    x0fPlatformDetails\t..JVM: (\d[^,]*), [^,]*, Oracle Corporation, OS: Linux, (\d\.[\d
    .]+)[^,]*, ([\w_-]+).*\x0fProviderVersion\t..(\d[\w._-]*)|s p/ActiveMQ OpenWire
    transport/ v/$4/ i/Java $1; arch: $3/ o/Linux $2/ cpe:/a:apache:activemq:$4/ cpe:/o:
    linux:linux_kernel:$2/a
softmatch apachemq m|^\0\0..\x01ActiveMQ\0| p/ActiveMQ OpenWire transport/


# Microsoft ActiveSync Version 3.7 Build 3083 (It's used for syncing
# my ipaq it disappears when you remove the ipaq.)
match activesync m|^.\0\x01\0[^\0]\0[^\0]\0[^\0]\0[^\0]\0[^\0]\0.*\0\0\0$|s p/Microsoft
    ActiveSync/ o/Windows/ cpe:/a:microsoft:activesync/ cpe:/o:microsoft:windows/a
```

```
match activesync m|^\(\0\0\0\x02\0\0\0\x03\0\0\0\+\0\0\x003\0\0\0\0\0\0\x04\0\0‘\x01
    \0\0\xff\0\0\0\0\0\0\0\0\0\0$|s p/Citrix ActiveSync/ o/Windows/ cpe:/o:microsoft:
    windows/a

match adabas-d m|^Adabas D Remote Control Server Version ([\d.]+) Date [\d-]+ \(key is
    [0-9a-f]+\)\r\nOK> | p/Adabas D database remote control/ v/$1/

match adobe-crossdomain m|^<cross-domain-policy><allow-access-from domain='([^']*)' to-
    ports='([^']*)' /></cross-domain-policy>\0$| p/Adobe cross-domain policy/ i/domain: $1
    ; ports: $2/
# Missing trailing \0? Was like that in the submission.
match adobe-crossdomain m|^<cross-domain-policy><allow-access-from domain=\"([^\"]*)\" to
    -ports=\"([^\"]*)\"/></cross-domain-policy>$| p/Adobe cross-domain policy/ i/domain:
    $1; ports: $2/
match adobe-crossdomain m|^<\?xml version=\"1\.0\"\?>\r\n<cross-domain-policy>\r\n    <
    site-control permitted-cross-domain-policies=\"master-only\"/>\r\n    <allow-access-
    from domain=\"\*\" to-ports=\"59160\"/>\r\n</cross-domain-policy>\0| p/Konica Minolta
    printer cross-domain-policy/
# playbrassmonkey.com
match adobe-crossdomain m|^<\?xml version=\"1\.0\"\?><cross-domain-policy><allow-access-
    from domain=\"\*\" to-ports=\"1008-49151\" /></cross-domain-policy>\0$| p/Brass Monkey
     cross-domain-policy/
match adobe-crossdomain m|^<\?xml version="1\.0"\?>\r\n<!DOCTYPE cross-domain-policy
    SYSTEM "http://www\.adobe\.com/xml/dtds/cross-domain-policy\.dtd">\r\n<cross-domain-
    policy>\r\n <site-control permitted-cross-domain-policies="master-only"/>\r\n <allow-
    access-from domain="www\.facebook\.com" to-ports="443" />\r\n</cross-domain-policy>\r\
    n| p/Facebook cross-domain policy/
softmatch adobe-crossdomain m|^<\?xml version=\"1\.0\"\?>.*<cross-domain-policy>|s

match afsmain m|^\+Welcome to Ability FTP Server \(Admin\)\. \[20500\]\r\n| p/Code-
    Crafters Ability FTP Server afsmain admin/ o/Windows/ cpe:/a:code-crafters:
    ability_ftp_server/ cpe:/o:microsoft:windows/a

match airserv-ng m|^\x05\0\0\x01.\0\0\0\0....\xff\xff\xff.\0\0\0\0\0\0\0.\0\0\0\0\0\x0fB@
    \0\0\0.\x80\0\0\0\xff\xff\xff\xff\xff\xff|s p/airserv-ng/ cpe:/a:aircrack-ng:airserv-
    ng/

match altiris-agent m|^<\0r\0e\0s\0p\0o\0n\0s\0e\0>\0C\0o\0n\0n\0e\0c\0t\0e\0d\0 \0t\0o\0
    [\0\d.]*<\0/\0r\0e\0s\0p\0o\0n\0s\0e\0>\0$| p/Altiris remote monitoring agent/


< ... >
```

1.3.6 Add new service to nmap-service-probes

Let's start simple TCP echo server by the following C++ code:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <map>

#define PORT 65100

#define BACKLOG 5
#define BUFFER_SIZE 1000
#define FLAGS 0

// Коллекция для хранения пар значений:
// сокет + идентификатор потока
```

```
20 std::map<int, pthread_t> threads;
21 // Серверный сокет
22 int serverSocket;
23
24 // Обработчик сигнала прерывания корректное( завершение приложения)
25 void signalHandler(int sig);
26 // Обработчик клиентского потока
27 void* clientExecutor(void* clientSocket);
28 // Функция считывания строки символов с клиента
29 int readLine(int socket, char* buffer, int bufferSize, int flags);
30 // Функция отправки строки символов клиенту
31 int sendLine(int socket, char* buffer, int flags);
32 // Корректное закрытие сокета
33 void closeSocket(int socket);
34 // Завершение работы клиентского потока
35 void destroyClient(int socket);
36
37 int main(int argc, char** argv) {
38   int port = PORT;
39   if(argc < 2)
40     printf("Using default port: %d.\n", port);
41   else
42     port = atoi(argv[1]);
43
44   // Создание серверного сокета
45   serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
46   if(serverSocket < 0) {
47     perror("It's impossible to create socket");
48     return 0x1;
49   }
50
51   printf("Server socket %d created.\n", serverSocket);
52
53   // Структура, задающая адресные характеристики
54   struct sockaddr_in info;
55   info.sin_family = AF_INET;
56   info.sin_port = htons(port);
57   info.sin_addr.s_addr = htonl(INADDR_ANY);
58
59   // Биндим сервер на определенный адрес
60   int serverBind  = bind(serverSocket, (struct sockaddr *) &info, sizeof(info));
61   if(serverBind < 0) {
62     perror("It's impossible to bind socket");
63     return 0x2;
64   }
65
66   // Слушаем сокет
67   int serverListen = listen(serverSocket, BACKLOG);
68   if(serverListen != 0) {
69     perror("It's impossible to listen socket");
70     return 0x3;
71   }
72
73   // Обработка прерывания для корректного завершения приложения
74   signal(SIGINT, signalHandler);
75
76   printf("Wait clients.\n");
77
78   while(1) {
79     // Ждем подключения клиентов
80     int clientSocket = accept(serverSocket, NULL, NULL);
81
82     // Пробуем создать поток обработки клиентских сообщений
83     pthread_t thread;
84     int result = pthread_create(&thread, NULL, clientExecutor, (void *) &clientSocket);
85     if(result) {
```

```
86        perror("It's impossible to create new thread");
87        closeSocket(clientSocket);
88      }
89
90      // Добавляем в коллекцию пару значений: сокет + идентификатор потока
91      threads.insert(std::pair<int, pthread_t>(clientSocket, thread));
92    }
93
94    return 0x0;
95  }
96
97  void signalHandler(int sig) {
98    // Для всех элементов коллекции
99    for(std::map<int, pthread_t>::iterator current = threads.begin(); current != threads.
        end(); ++current) {
100     printf("Try to finish client with socket %d\n", current->first);
101     // Закрываем клиентские сокеты
102     closeSocket(current->first);
103     printf("Client socket %d closed.\n", current->first);
104   }
105
106   // Закрываем серверный сокет
107   closeSocket(serverSocket);
108   printf("Server socket %d closed.\n", serverSocket);
109
110   exit(0x0);
111 }
112
113 void* clientExecutor(void* socket) {
114   int clientSocket = *((int*) socket);
115
116   printf("Client thread with socket %d created.\n", clientSocket);
117
118   char buffer[BUFFER_SIZE];
119   while(1) {
120     // Ожидаем прибытия строки
121     int result = readLine(clientSocket, buffer, BUFFER_SIZE, FLAGS);
122     if(result < 0)
123       destroyClient(clientSocket);
124
125     if(strlen(buffer) <= 1)
126       destroyClient(clientSocket);
127
128     printf("Client message: %s\n", buffer);
129
130     // Отправляем строку назад
131     result = sendLine(clientSocket, buffer, FLAGS);
132     if(result < 0)
133       destroyClient(clientSocket);
134   }
135 }
136
137 int readLine(int socket, char* buffer, int bufferSize, int flags) {
138   // Очищаем буфер
139   bzero(buffer, bufferSize);
140
141   char resolvedSymbol = ' ';
142   for(int index = 0; index < BUFFER_SIZE; ++index) {
143     // Считываем по одному символу
144     int readSize = recv(socket, &resolvedSymbol, 1, flags);
145     if(readSize <= 0)
146         return -1;
147     else if(resolvedSymbol == '\n')
148         break;
149     else if(resolvedSymbol != '\r')
150         buffer[index] = resolvedSymbol;
```

```
151   }
152
153   return 0x0;
154 }
155
156 int sendLine(int socket, char* buffer, int flags) {
157   unsigned int length = strlen(buffer);
158
159   // Перед отправкой сообщения добавляем в конец перевод строки
160   if(length == 0)
161     return -1;
162   else if(buffer[length - 1] != '\n') {
163     if(length >= BUFFER_SIZE)
164       return -1;
165     else
166       buffer[length] = '\n';
167
168   }
169
170   length = strlen(buffer);
171
172   // Отправляем строку клиенту
173   int result = send(socket, buffer, length, flags);
174   return result;
175 }
176
177 void closeSocket(int socket) {
178   // Завершение работы сокета
179   int socketShutdown = shutdown(socket, SHUT_RDWR);
180       if(socketShutdown != 0)
181     perror("It's impossible to shutdown socket");
182
183   // Закрытие сокета
184   int socketClose = close(socket);
185       if(socketClose != 0)
186     perror("It's impossible to close socket");
187 }
188
189 void destroyClient(int socket) {
190   printf("It's impossible to receive message from client or send message to client.\n");
191   // Завершение работы сокета
192   closeSocket(socket);
193   // Удаление пары значений из коллекции по ключю
194   threads.erase(socket);
195   printf("Client socket %d closed.\n", socket);
196   // Завершение работы потока
197   pthread_exit(NULL);
198 }
```

Compile and run the server on port 65100:

```
root@kali:~# mkdir temp
root@kali:~# cd temp/
root@kali:~/temp# touch server.cpp
root@kali:~/temp# gedit server.cpp
^C
root@kali:~/temp# g++ -o server -pthread server.cpp
root@kali:~/temp# ./server
Using default port: 65100.
Server socket 3 created.
Wait clients.
```

Let's try to check **nmap** result before changes into nmap-service-probes file:

```
root@kali:~/temp# nmap -sV -p 65100 localhost

Starting Nmap 7.60 ( https://nmap.org ) at 2017-11-05 16:52 EST
```

```
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000028s latency).
Other addresses for localhost (not scanned): ::1

PORT      STATE SERVICE VERSION
65100/tcp open  unknown
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port65100-TCP:V=7.60%I=7%D=11/5%Time=59FF8815%P=x86_64-pc-linux-gnu%r(G
SF:etRequest,F,"GET\x20/\x20HTTP/1\.0\n")%r(HTTPOptions,13,"OPTIONS\x20/\x
SF:20HTTP/1\.0\n")%r(RTSPRequest,13,"OPTIONS\x20/\x20RTSP/1\.0\n")%r(Help,
SF:5,"HELP\n")%r(SSLSessionReq,3,"\x16\x03\n")%r(TLSSessionReq,3,"\x16\x03
SF:\n")%r(FourOhFourRequest,32,"GET\x20/nice%20ports%2C/Tri%6Eity\.txt%2eb
SF:ak\x20HTTP/1\.0\n")%r(LPDString,9,"\x01default\n")%r(LDAPSearchReq,3,"0
SF:\x84\n")%r(SIPOptions,D3,"OPTIONS\x20sip:nm\x20SIP/2\.0\nVia:\x20SIP/2\
SF:.0/TCP\x20nm;branch=foo\nFrom:\x20<sip:nm@nm>;tag=root\nTo:\x20<sip:nm2
SF:@nm2>\nCall-ID:\x2050000\nCSeq:\x2042\x20OPTIONS\nMax-Forwards:\x2070\n
SF:Content-Length:\x200\nContact:\x20<sip:nm@nm>\nAccept:\x20application/s
SF:dp\n");

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 105.15 seconds
```

The results indicate that it was not possible to determine the type of port and application.

The echo server displays information about requests coming to it. This illustrates the operating principle of the **nmap** utility: various requests are sent and depending on the response, the type of protocol is determined.

```
root@kali:~/temp# ./server
Using default port: 65100.
Server socket 3 created.
Wait clients.
Client thread with socket 4 created.
It's impossible to receive message from client or send message to client.
Client socket 4 closed.
Client thread with socket 5 created.
Client message: GET / HTTP/1.0
It's impossible to receive message from client or send message to client.
Client socket 5 closed.
Client thread with socket 4 created.
Client message: OPTIONS / HTTP/1.0
It's impossible to receive message from client or send message to client.
Client socket 4 closed.
Client thread with socket 5 created.
Client message: OPTIONS / RTSP/1.0
< ... >
```

The following code was added to the **nmap-service-probes** file, which will help **nmap** determine the type and version of the application:

```
< ... >

############################NEXT PROBE############################
# My TCP server.
Probe TCP my-tcp-server-ver q|version\r\n|
rarity 9
ports 65100
match echo m|^1\.0\.0$| p/My TCP Server/ v/1.0.0/
```

After this changes **nmap** successfully recognize the type of protocol:

```
root@kali:~/temp# nmap -sV -p 65100 localhost

Starting Nmap 7.60 ( https://nmap.org ) at 2017-11-05 16:56 EST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000027s latency).
Other addresses for localhost (not scanned): ::1
```

```
7
8  PORT      STATE SERVICE VERSION
9  65100/tcp open  echo    My TCP Server 1.0.0
10
11 Service detection performed. Please report any incorrect results at https://nmap.org/
       submit/ .
12 Nmap done: 1 IP address (1 host up) scanned in 6.51 seconds
```

1.3.7 Study nmap stages and modes using Wireshark

Consider the network interaction of the utility nmap in the Wireshark program. Let's try to get 5 most used ports of Metasploitable 2 OS:

```
root@kali:~# nmap -sV -top-ports 5 192.168.56.101
```

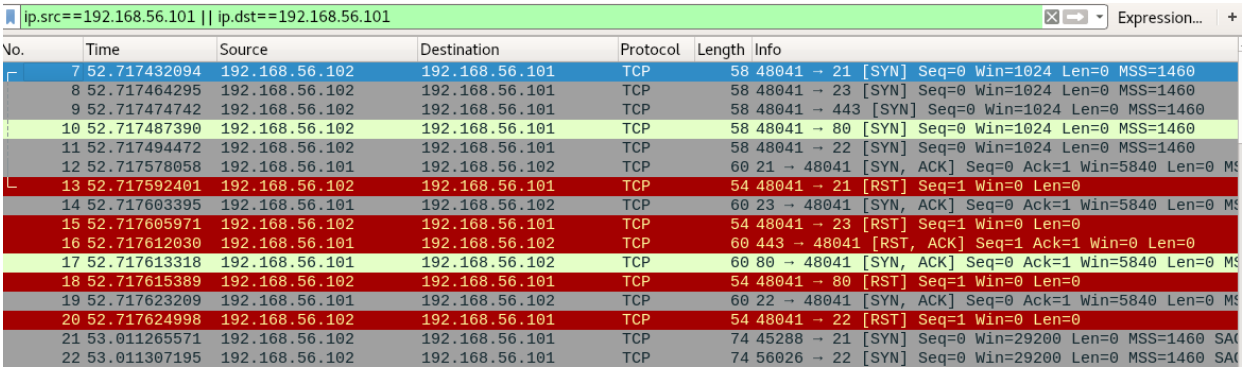

Рис. 1.4: Trying to get 5 most used ports

Nmap tries to establish TCP connections with a lot of ports on the target system. if a packet with the RST flag is returned, the namp concludes that the port is closed. If no response is received, then the port is filtered or does not exist.

Let's try to get information about the TCP server, that is running on port 65100:

```
msfadmin@metasploitable:~$ nmap -sV -p 65100 192.168.56.102
```

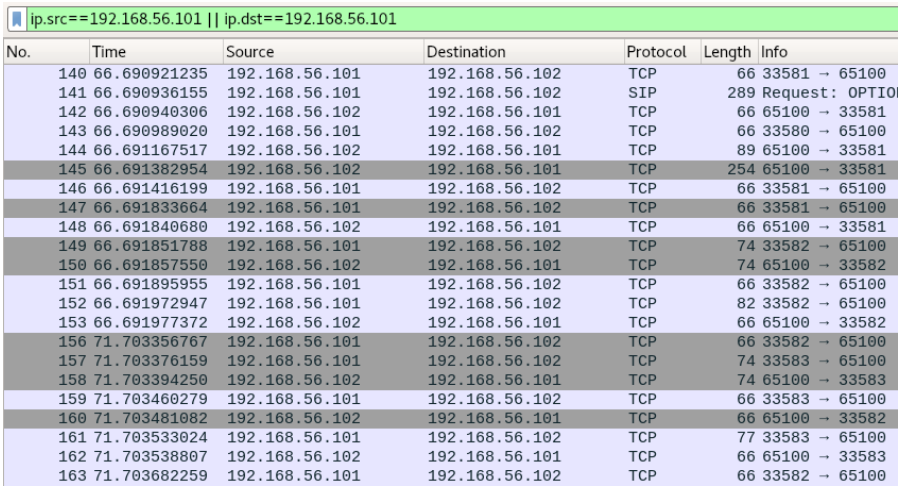

Рис. 1.5: Trying to get information about port 65100

This illustrates the operating principle of the **nmap** utility: various requests are sent and depending on the response, the type of protocol is determined.

1.3.8 Output to xml-format file

The result of the operation can be represented in the XML format:

```
root@kali:~/temp# nmap -sV -p 65100 -oX - scanme.nmap.org localhost
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nmaprun>
<?xml-stylesheet href="file:///usr/bin/../share/nmap/nmap.xsl" type="text/xsl"?>
<!-- Nmap 7.60 scan initiated Sun Nov  5 17:03:42 2017 as: nmap -sV -p 65100 -oX - scanme
    .nmap.org localhost -->
<nmaprun scanner="nmap" args="nmap -sV -p 65100 -oX - scanme.nmap.org localhost" start="
    1509919422" startstr="Sun Nov  5 17:03:42 2017" version="7.60" xmloutputversion="1.04"
    >
<scaninfo type="syn" protocol="tcp" numservices="1" services="65100"/>
<verbose level="0"/>
<debugging level="0"/>
Failed to resolve "scanme.nmap.org".
<host starttime="1509919422" endtime="1509919429"><status state="up" reason="localhost-
    response" reason_ttl="0"/>
<address addr="127.0.0.1" addrtype="ipv4"/>
<hostnames>
<hostname name="localhost" type="user"/>
<hostname name="localhost" type="PTR"/>
</hostnames>
<ports><port protocol="tcp" portid="65100"><state state="open" reason="syn-ack"
    reason_ttl="64"/><service name="echo" product="My TCP Server" version="1.0.0" method="
    probed" conf="10"/></port>
</ports>
<times srtt="32" rttvar="5000" to="100000"/>
</host>
<runstats><finished time="1509919429" timestr="Sun Nov  5 17:03:49 2017" elapsed="6.53"
    summary="Nmap done at Sun Nov  5 17:03:49 2017; 1 IP address (1 host up) scanned in
    6.53 seconds" exit="success"/><hosts up="1" down="0" total="1"/>
</runstats>
</nmaprun>
```

1.3.9 Perform VM Metasploitable2 scanning using db_nmap from metasploitframework

We can use the db_nmap command to run Nmap against our targets and our scan results would than be stored automatically in our database. However, if you also wish to import the scan results into another application or framework later on, you will likely want to export the scan results in XML format.

```
msf > db_nmap -v -sV 192.168.56.101
[*] Nmap: Starting Nmap 7.60 ( https://nmap.org ) at 2017-11-05 17:34 EST
[*] Nmap: NSE: Loaded 42 scripts for scanning.
[*] Nmap: Initiating ARP Ping Scan at 17:34
[*] Nmap: Scanning 192.168.56.101 [1 port]
[*] Nmap: Completed ARP Ping Scan at 17:34, 0.23s elapsed (1 total hosts)
[*] Nmap: Initiating Parallel DNS resolution of 1 host. at 17:34
[*] Nmap: Completed Parallel DNS resolution of 1 host. at 17:35, 13.00s elapsed
[*] Nmap: Initiating SYN Stealth Scan at 17:35
[*] Nmap: Scanning 192.168.56.101 [1000 ports]
[*] Nmap: Discovered open port 53/tcp on 192.168.56.101
[*] Nmap: Discovered open port 80/tcp on 192.168.56.101
[*] Nmap: Discovered open port 111/tcp on 192.168.56.101
[*] Nmap: Discovered open port 22/tcp on 192.168.56.101
[*] Nmap: Discovered open port 21/tcp on 192.168.56.101
[*] Nmap: Discovered open port 445/tcp on 192.168.56.101
[*] Nmap: Discovered open port 23/tcp on 192.168.56.101
[*] Nmap: Discovered open port 139/tcp on 192.168.56.101
[*] Nmap: Discovered open port 25/tcp on 192.168.56.101
[*] Nmap: Discovered open port 3306/tcp on 192.168.56.101
[*] Nmap: Discovered open port 5900/tcp on 192.168.56.101
[*] Nmap: Discovered open port 1099/tcp on 192.168.56.101
[*] Nmap: Discovered open port 8009/tcp on 192.168.56.101
[*] Nmap: Discovered open port 514/tcp on 192.168.56.101
```

```
[*] Nmap: Discovered open port 513/tcp on 192.168.56.101
[*] Nmap: Discovered open port 2121/tcp on 192.168.56.101
[*] Nmap: Discovered open port 5432/tcp on 192.168.56.101
[*] Nmap: Discovered open port 2049/tcp on 192.168.56.101
[*] Nmap: Discovered open port 8180/tcp on 192.168.56.101
[*] Nmap: Discovered open port 512/tcp on 192.168.56.101
[*] Nmap: Discovered open port 6000/tcp on 192.168.56.101
[*] Nmap: Discovered open port 6667/tcp on 192.168.56.101
[*] Nmap: Discovered open port 1524/tcp on 192.168.56.101
[*] Nmap: Completed SYN Stealth Scan at 17:35, 1.25s elapsed (1000 total ports)
[*] Nmap: Initiating Service scan at 17:35
[*] Nmap: Scanning 23 services on 192.168.56.101
[*] Nmap: Completed Service scan at 17:35, 11.11s elapsed (23 services on 1 host)
[*] Nmap: NSE: Script scanning 192.168.56.101.
[*] Nmap: Initiating NSE at 17:35
[*] Nmap: Completed NSE at 17:35, 0.08s elapsed
[*] Nmap: Initiating NSE at 17:35
[*] Nmap: Completed NSE at 17:35, 0.01s elapsed
[*] Nmap: Nmap scan report for 192.168.56.101
[*] Nmap: Host is up (0.00028s latency).
[*] Nmap: Not shown: 977 closed ports
[*] Nmap: PORT     STATE SERVICE     VERSION
[*] Nmap: 21/tcp   open  ftp         vsftpd 2.3.4
[*] Nmap: 22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
[*] Nmap: 23/tcp   open  telnet      Linux telnetd
[*] Nmap: 25/tcp   open  smtp        Postfix smtpd
[*] Nmap: 53/tcp   open  domain      ISC BIND 9.4.2
[*] Nmap: 80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
[*] Nmap: 111/tcp  open  rpcbind     2 (RPC #100000)
[*] Nmap: 139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
[*] Nmap: 445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
[*] Nmap: 512/tcp  open  exec        netkit-rsh rexecd
[*] Nmap: 513/tcp  open  login?
[*] Nmap: 514/tcp  open  shell       Netkit rshd
[*] Nmap: 1099/tcp open  rmiregistry GNU Classpath grmiregistry
[*] Nmap: 1524/tcp open  shell       Metasploitable root shell
[*] Nmap: 2049/tcp open  nfs         2-4 (RPC #100003)
[*] Nmap: 2121/tcp open  ftp         ProFTPD 1.3.1
[*] Nmap: 3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
[*] Nmap: 5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
[*] Nmap: 5900/tcp open  vnc         VNC (protocol 3.3)
[*] Nmap: 6000/tcp open  X11         (access denied)
[*] Nmap: 6667/tcp open  irc         UnrealIRCd
[*] Nmap: 8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
[*] Nmap: 8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
[*] Nmap: MAC Address: 08:00:27:7E:E4:CC (Oracle VirtualBox virtual NIC)
[*] Nmap: Service Info: Hosts:  metasploitable.localdomain, localhost, irc.Metasploitable
    .LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
[*] Nmap: Read data files from: /usr/bin/../share/nmap
[*] Nmap: Service detection performed. Please report any incorrect results at https://
    nmap.org/submit/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 26.09 seconds
[*] Nmap: Raw packets sent: 1001 (44.028KB) | Rcvd: 1001 (40.120KB)
```

1.3.10 Get some records from nmap-service-probes and describe them

`Probe <protocol> <probename> <probestring>`

The Probe directive tells Nmap what string to send to recognize various services. All of the directives discussed later operate on the most recent Probe statement. The arguments are as follows:

- <**protocol**> – This must be either TCP or UDP. Nmap only uses probes that match the protocol of the service it is trying to scan.
- <**probename**> – This is a plain English name for the probe. It is used in service fingerprints to describe which probes elicited responses.

- <**probestring**> – Tells Nmap what to send.

```
match <service> <pattern> [<versioninfo>]
```

The match directive tells Nmap how to recognize services based on responses to the string sent by the previous Probe directive. The arguments to this directive follow:

- <**service**> – This is simply the service name that the pattern matches.
- <**pattern**> – This pattern is used to determine whether the response received matches the service given in the previous parameter.
- <**versioninfo**> – actually contains several optional fields. Each field begins with an identifying letter (such as h for "hostname"). Next comes a delimiter character which the signature writer chooses.

```
rarity <value between 1 and 9>
```

The rarity directive roughly corresponds to how infrequently this probe can be expected to return useful results. The higher the number, the more rare the probe is considered and the less likely it is to be tried against a service.

```
ports <portlist>
```

This line tells Nmap what ports the services identified by this probe are commonly found on. It should only be used once within each Probe section.

1.3.11 Choose one Nmap Script and describe it

The Nmap Scripting Engine (NSE) is one of Nmap's most powerful and flexible features. It allows users to write (and share) simple scripts to automate a wide variety of networking tasks. Those scripts are then executed in parallel with the speed and efficiency you expect from Nmap. Users can rely on the growing and diverse set of scripts distributed with Nmap, or write their own to meet custom needs.

The following script starts all unit tests for the nmap utility and located at /usr/share/nmap/scripts/unittest.nse:

```
1  local stdnse = require "stdnse"
2  local unittest = require "unittest"
3
4  description = [[
5  Runs unit tests on all NSE libraries.
6  ]]
7
8  ---
9  -- @args unittest.run Run tests. Causes <code>unittest.testing()</code> to
10 --                    return true.
11 --
12 -- @args unittest.tests Run tests from only these libraries (defaults to all)
13 --
14 -- @usage
15 -- nmap --script unittest --script-args unittest.run
16 --
17 -- @output
18 -- Pre-scan script results:
19 -- | unittest:
20 -- |_ All tests passed
21
22 author = "Daniel Miller"
23
24 license = "Same as Nmap--See https://nmap.org/book/man-legal.html"
25
26 categories = {"safe"}
27
28
29 prerule = unittest.testing
30
31 action = function()
32   local libs = stdnse.get_script_args("unittest.tests")
33   local result
```

```
34    if libs then
35      result = unittest.run_tests(libs)
36    else
37      result = unittest.run_tests()
38    end
39    if #result == 0 then
40      return "All tests passed"
41    else
42      return result
43    end
44  end
```

This script contains the following fields:

- **description** – describes what a script is testing for and any important notes the user should be aware of.
- **author** – contains the script authors' names and can also contain contact information.
- **license** – helps ensure that we have legal permission to distribute all the scripts which come with Nmap.
- **categories** – defines one or more categories to which a script belongs.
- **prerule** – determines whether a script should be run against a target. Prerule run once, before any hosts are scanned, during the script pre-scanning phase.
- **action** – contains all of the instructions to be executed when the script's prerule, portrule, hostrule or postrule triggers.

1.4 Conclusion

Nmap is a free and open source utility for network discovery and security auditing. In this laboratory work, the main features of this utility were studied, and experiments were performed to illustrate its power. All these experiments show that remote attackers can get a lot of information about the system, so to ensure security, you must use a firewall.