# Bluffing Network Scan Tools

## What you see may not be what you get

**Date of writing**: 10/2015

**Author**: Emeric Nasi – emeric.nasi[at]sevagas.com

**Website**: http://www.sevagas.com/

**Note**: Pentesting / Network security basics are recommended to understand this paper.

# 1. Introduction

## 1.1. Relying on automatic tools

I have often seen tutorials or even pro pentesters relying too much, if not uniquely on automatic scanning tools. It may be due to lack of knowledge, or more often due to lack of available time. Obviously when you have 5 week days to complete a full corporate pentest, you can only do your best, and it won't be perfect!

Anyway I just wanted to write a little something to remind that automatic tools results are always interpretation of incoming data. Tools expect a certain behavior from systems, and will make some assumptions. If you do not know this, you may be fooled by false positives or worse loose your valuable time!

Just a quick example, when you successfully ping a machine, so you assume it's alive. But in fact, it just means you received and ICMP Echo Reply packet in answer to sending an ICMP Echo Request. This echo reply could have been send by another machine than the targeted one. It can be part of a tarpit strategy!

Now let's focus on some major features of security scan tools:

- OS fingerprinting
- Port Scanning
- Banner grabbing

## 1.2. Set up

I've ran my tests using two virtual machines running both on Linux. The targeted host is an Ubuntu 14.04 Linux with kernel 3.13.0 and has IP 192.168.0.16.

I targeted this host with free widely known scanning tools which are:

- Nmap (https://nmap.org/)
- OpenVas *(http://www.openvas.org/)*
- Nikto (http://www.cirt.net/Nikto2)

All the trick code in this article are summed up in a bash script in section §6 as an Annex at the end of the document.  You can copy past this code in a bash, run it as root, modify it and have fun!

## 2. OS Fingerprinting

### 2.1. Behind the scene

OS Fingerprinting, whether it is active (ex. using Nmap) or passive (ex. using p0f) relies on how the target IP stack behaves in a way that distinguishes it from other systems.

Automatic tools will compare several behaviors and packets fields to a signature database. Automatic tools will assume that IP stack behavior will always be the same for a given OS. This is indeed normally the case, unless you modify it.

### 2.2. How it can be tricked

OS fingerprinting can be tricked by modifying the behavior of the system TCP/IP stack.

One of the values which are looked at by OS fingerprinting tools is an IP default Time To Leave (TTL) field in the packet. It is also very easy to modify on a Linux Box. The default TTL value on Linux is generally set to 64. As you can check by doing:

```
#  sysctl net.ipv4.ip_default_ttl
net.ipv4.ip_default_ttl = 64
```

You can change this value to 128 which is, among others, Microsoft Windows default TTL value.

```
# sysctl -w net.ipv4.ip_default_ttl=128
```

Let's see how Nmap and OpenVas react to this simple command.

<u>**Example 1: Nmap OS detection**</u>

The -O option is used by Nmap to fingerprint. The way Nmap fingerprints OS is fully explained at https://nmap.org/book/man-os-detection.html

```
# nmap -O 192.168.0.16

Starting Nmap 6.00 ( http://nmap.org ) at 2015-09-27 18:13 CEST
Nmap scan report for 192.168.0.16
Host is up (0.00042s latency).
Not shown: 995 closed ports
PORT     STATE    SERVICE
21/tcp   filtered ftp
...
MAC Address: xxxxxxxxxxxxxxxxxxxxxx
No exact OS matches for host (If you know what OS is running on it, see http://nmap.org/submit/ ).
...
```

Nmap builds signatures based on various behaviors it expects from the target IP stack.

In this case we can see we came up with something outside of Nmap signature base. However we just change one value, the TTL. With more kernel tweaking, we could totally fake another OS signature on a Linux Box

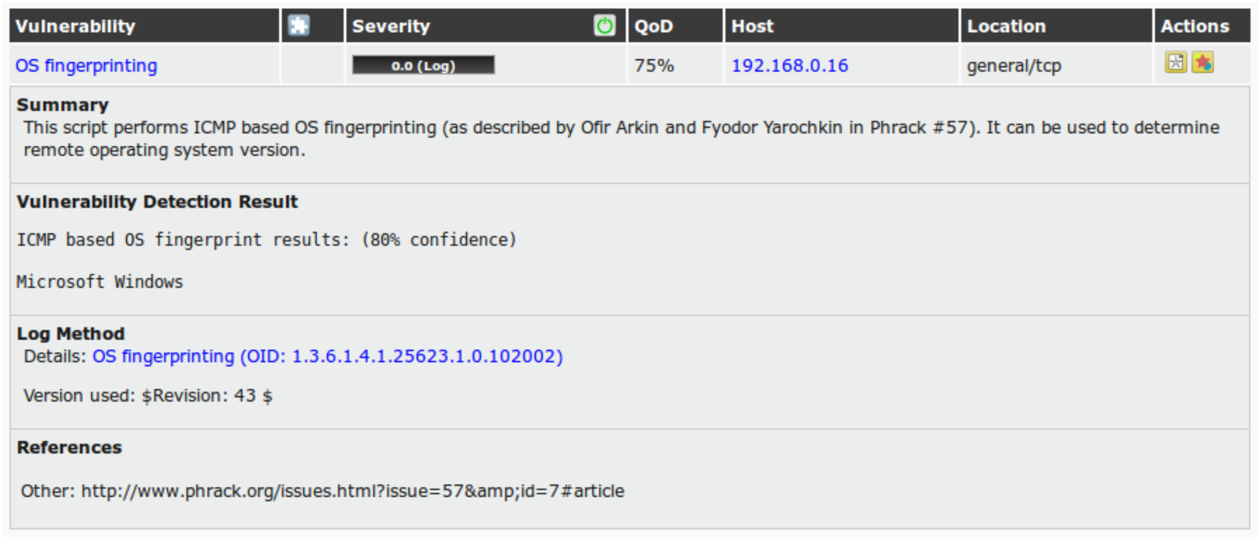
We can use the –osscan-guess option to see what Nmap thinks our machine is most probably:

```
# nmap -O --osscan-guess 192.168.0.16

Starting Nmap 6.00 ( http://nmap.org ) at 2015-09-27 18:17 CEST
Nmap scan report for 192.168.0.16
Host is up (0.00042s latency).
....
MAC Address: xxxxxxxxxxxxxxxxxxxxx
Device type: WAP|media device|general purpose|webcam|specialized|printer|PBX
Running (JUST GUESSING): Netgear embedded (92%), Western Digital embedded (92%), Linux
2.6.X|3.X|2.4.X (89%), AXIS Linux 2.6.X (88%), Crestron 2-Series (87%), Lexmark embedded (87%), Vodavi
embedded (86%), Comtrend embedded (86%)
...
```

**Example 2: OpenVAS OS detection**

The same test was done with OpenVas open source vulnerability scanner. The default scan does also run OS fingerprinting mechanism, based on ICMP fingerprinting.



| Vulnerability | | Severity | QoD | Host | Location | Actions |
|---|---|---|---|---|---|---|
| OS fingerprinting | | 0.0 (Log) | 75% | 192.168.0.16 | general/tcp | |

**Summary**
This script performs ICMP based OS fingerprinting (as described by Ofir Arkin and Fyodor Yarochkin in Phrack #57). It can be used to determine remote operating system version.

**Vulnerability Detection Result**

```
ICMP based OS fingerprint results: (80% confidence)

Microsoft Windows
```

**Log Method**
Details: OS fingerprinting (OID: 1.3.6.1.4.1.25623.1.0.102002)

Version used: $Revision: 43 $

**References**

Other: http://www.phrack.org/issues.html?issue=57&amp;id=7#article

As you can see, OpenVas is much more confident in its capacity then Nmap and gives a unique result: Our host is running Microsoft Windows (80% confidence).

# 3. Port scanning

## 3.1. Behind the scene

Port scanning also relies on IP stack behavior, but this time on expected normal implementation of the RFC (which is not always the case on all OS but let's not start on this topic!).

The most common scans used by tools are Syn scan and TCP connect scan. There are also other variant of "exotic" scans playing with various possibilities of the TCP flag. Here is the normal behavior, expected by port scan tools:

If the port is opened:

```
Scanner   –--- SYN  –--->   Target

Scanner   <– SYN,ACK --   Target
```

If the port is closed:

```
Scanner   –--- SYN  –--->   Target

Scanner   <–   RST –-----   Target
```

For any other behavior, the port is considered filtered, behind a firewall.

Concerning other types of scans, you can find more information on this post I wrote a few years ago: http://www.sevagas.com/?Iptables-firewall-versus-nmap-and,31

## 3.2. How to detect it

Knowing how it works makes it possible to build a defense. In the code below we use iptables to log various scan attempts, including exotic scans.

```
  iptables -A INPUT -p tcp --tcp-flags ALL FIN,PSH,URG -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix "Firewall>XMAS scan "

   iptables -A INPUT -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix "Firewall>XMAS-PSH scan "

   iptables -A INPUT -p tcp --tcp-flags ALL ALL -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix "Firewall>XMAS-ALL scan "
   iptables -A INPUT -p tcp --tcp-flags ALL FIN -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix
"Firewall>FIN scan "

   iptables -A INPUT -p tcp --tcp-flags SYN,RST SYN,RST -m limit --limit 6/h --limit-burst 1 -j LOG --log-
prefix "Firewall>SYN/RST scan "

   iptables -A INPUT -p tcp --tcp-flags SYN,FIN SYN,FIN -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix
"Firewall>SYN/FIN scan "

   iptables -A INPUT -p tcp --tcp-flags ALL NONE -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix
"Firewall>Null scan "

# Prevent Syn port scan (will detect syn on 5 different port in less than 3 sec)

  iptables -A INPUT -m psd -p tcp -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix "Firewall>Syn scan
detected "
```

Note that the previous lines will only logs the scan, they will not prevent it. The goal is more to show that by knowing the TCP flags used by ports scaners, we can detect them.

Concerning Syn and TCP connect scan, since they correspond to normal TCP connection behavior, they are much more difficult to detect. One way, as seen above, is to use the iptables psd module which can detect a certain amount of connection to different ports from the same IP in a given lapse of time.

This will not work is the scan is really slow. Another possibility is to “trap” some ports which are never used on your systems. For example port 445 (SMB) on a Linux webserver or port 23 (Telnet) anywhere. In these cases you are sure a hit on the port is a malicious activity. This is easy to do using iptables.

### 3.3. How to trick port scanning

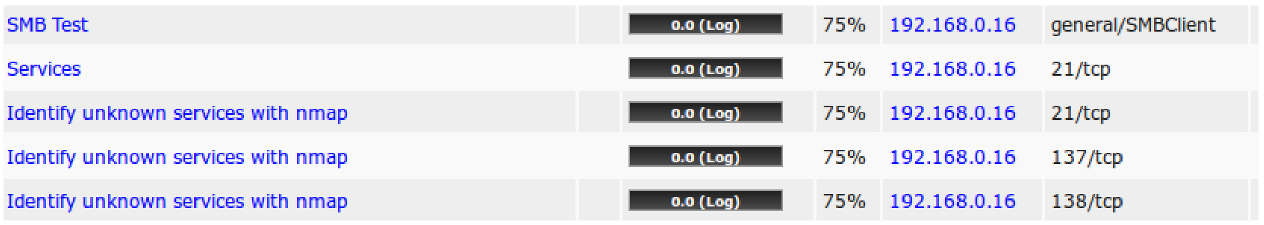
An easy way to trick Syn and TCP connect port scanning is to pretend some or all ports are opened. This can be done by responding Syn,Ack packet to any Syn query. This way, closed and filtered ports can be thought opened.

We can do that using iptables TARPIT target:

```
        iptables -A  fw-aggressive -p tcp  --destination-port 21 -j TARPIT
        iptables -A  fw-aggressive -p tcp  --destination-port 137:139 -j TARPIT
        iptables -A  fw-aggressive -p tcp  --destination-port 445 -j TARPIT
```

The TARPIT target will answer Syn/ACK packet to any Syn query on the filtered ports. This is useful to trick automatic port scanner, especially those used by fast spreading worms and botnets.

**Example 1: OpenVAS port scanning**



| | | | | | |
|---|---|---|---|---|---|
| SMB Test | | 0.0 (Log) | 75% | 192.168.0.16 | general/SMBClient |
| Services | | 0.0 (Log) | 75% | 192.168.0.16 | 21/tcp |
| Identify unknown services with nmap | | 0.0 (Log) | 75% | 192.168.0.16 | 21/tcp |
| Identify unknown services with nmap | | 0.0 (Log) | 75% | 192.168.0.16 | 137/tcp |
| Identify unknown services with nmap | | 0.0 (Log) | 75% | 192.168.0.16 | 138/tcp |

As you can see, this default OpenVas scan assumes that our rogue ports are opened. It fell into the tarpit. In reality, none of these ports are opened on the machine.

A well-crafted tarpit could lead vulnerability scanner to try to find vulnerabilities on the supposed services and also return false positives. This is the case for example when you implement a Honeypot. The Dionaea honeypot (http://dionaea.carnivore.it/) for example does a good job implementing false insecure SMB services (however default banner grabbing will detect Dionaea as a service if it is not modified).
This is a great way to loose a lot of time for the "attacker"…

**Example 2: Nmap port scanning**

The iptables TARPIT does not work against latest versions of Nmap

```
# nmap 192.168.0.16
...
Not shown: 995 closed ports
PORT     STATE    SERVICE
21/tcp   filtered ftp
139/tcp  filtered netbios-ssn
445/tcp  filtered microsoft-ds
...
```

We can see that despite the tarpit, Nmap detects that the ports are being filtered. On other versions, Nmap will declare the ports as Open. In fact it is possible to detect the use of iptables TARPIT. This is because the iptables TARPIT model returns a SYN/ACK packet which has singularities (for example, a max sized segment of 0, which is not normally the case). This is well explained in this link https://www.fishnetsecurity.com/6labs/blog/port-scanning-through-tarpits. It is thus possible to detect the usage of TARPIT iptables target for the scanner.

Anyway, even if Nmap can now bypass this simple use of TARPIT; It is possible to build a more sophisticated one which returns a Syn/Ack packet in a much more realistic fashion.

## 4. Banner Grabbing

### 4.1. Behind the scene

Banner grabbing process is pretty simple, the scanner will act this time on application layer. It will send a classic application request to the service, which will answer back. In the answer, the tools expect to find the service name and version.

### 4.2. How to trick it

The most common methods used to trick both port scanning and banner grabbing is to use honeypots or sophisticated tarpits applications.

If you want to test a quick demo, here is a nice little trick from Nick Marsh (see http://marc.info/?l=nmap-dev&m=142811327627471&w=2) to send HTTP application scanner to an infinite loophole.

Run the next command on the target machine

```
#  ncat -lkv -p 8080 --sh-exec "echo 'HTTP/1.1 200 OK\r\n'; cat /dev/urandom"
```

**Note**: This is not something to run on production system, it may eat up your CPU after several scan attempts!

**Example 1: Using Nmap**

A classic Nmap scan will just show the port is opened:

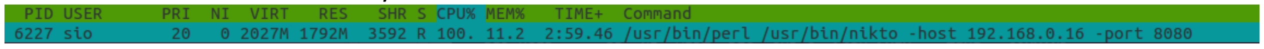
```
8080/tcp open     http-proxy
```

However it is another story for Nmap script which attempts banner grabbing.
Nmap -sC or Nmap -A options will result in Nmap trying to pull an infinit HTTP message. This makes Nmap run infinitely while consuming a lot of resources, and at one point, Nmap crashes.

**Example 2: Using Nikto**

```
nikto -host 192.168.0.16 -port 8080
- Nikto v2.1.4
---------------------------------------------------------------------------
+ Target IP:          192.168.0.16
+ Target Hostname:    192.168.0.16
+ Target Port:        8080
+ Start Time:         2015-09-28 09:34:03
---------------------------------------------------------------------------
+ Server: No banner retrieved
```

It is the same for Nikto. During the scan, while nothing happened, Nikto process was eating up all its host CPU resource and I had to finally kill it.



```
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 6227 sio        20   0 2027M 1792M  3592 R 100. 11.2  2:59.46 /usr/bin/perl /usr/bin/nikto -host 192.168.0.16 -port 8080
```

## 5. To Conclude

With easy to implement examples, we can see that it is possible to trick security tools. When running vulnerability scanners or any other kind of security tools, remember what is going on behind the scene. Remember that the tools answers are “assumptions” and not definitive answers.
The Security tools can also be prone to vulnerabilities when running which can result in the scanner being attacked by its target as we saw in the HTTP infinite tarpit case.

**Warning**: The methods in this article are all related to Security by Obscurity. It is generally recommended to avoid Security by Obscurity but rather use standard and strong methodology. However for some companies, Security By Obscurity is, when well used, only one of the security layers protecting the system.
In my opinion Security by Obscurity should only be used:
- If your system already has really solid security “by standards”
- If you are fully aware of the risks induced by your non-standards security method
- If both IT and Security team are aware of the implemented mechanism (or else you will “obscure” yourselfs..)

For me, experimenting on this topic is mainly about having some fun with my favorite Operating System!
Here are some interesting links if you want to go further:

http://www.sans.org/reading-room/whitepapers/tools/about-face-defending-organization-penetration-testing-teams-33553

http://labrea.sourceforge.net/Intro-History.html

https://nmap.org/book/man-os-detection.html

https://www.fishnetsecurity.com/6labs/blog/port-scanning-through-tarpits

http://www.netfilter.org/projects/patch-o-matic/pom-external.html

http://www.sevagas.com/?Iptables-firewall-versus-nmap-and

## 6. ANNEX: Demo Bash Script

```
-------------------- Begin script ---------------------------

#!/bin/bash

# Emeric NASI - www.sevagas.com
# Some fun tweeking using linux kernel and firewall to trick security scanning tools
# Run as root
#
# Note: This is not a defensive firewall script, do not use as is to protect your network ports!
#


# Rules to pretend ports are opened
false_opened_ports()
{
        iptables -N fw-aggressive # This chain  is used to process IP where we apply aggressive defence
measures
        iptables -A INPUT  -m comment --comment "Pretend ports are opened" --jump fw-aggressive
        iptables -A  fw-aggressive -p tcp  --destination-port 21 -j TARPIT
        iptables -A  fw-aggressive -p tcp  --destination-port 137:139 -j TARPIT
        iptables -A  fw-aggressive -p tcp  --destination-port 445 -j TARPIT
   # If you have DROP default output policy remember to:
        #iptables -A OUTPUT -p tcp --tcp-flags SYN,ACK SYN,ACK --source-port 21 -j ACCEPT
        #iptables -A OUTPUT -p tcp --tcp-flags SYN,ACK SYN,ACK --source-port 137:139 -j ACCEPT
        #iptables -A OUTPUT -p tcp --tcp-flags SYN,ACK SYN,ACK --source-port 445 -j ACCEPT
}


# Port scanning detection
detect_port_scan ()
{
   # Drop and Log various port scannings types (Fin, Null, Xmas tree, etc.)"
   iptables -A INPUT -p tcp --tcp-flags ALL FIN,PSH,URG -m limit --limit 6/h --limit-burst 1 -j LOG --log-
prefix "Firewall>XMAS scan "
   iptables -A INPUT -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG -m limit --limit 6/h --limit-burst 1 -j LOG --
log-prefix "Firewall>XMAS-PSH scan "
   iptables -A INPUT -p tcp --tcp-flags ALL ALL -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix
"Firewall>XMAS-ALL scan "
   iptables -A INPUT -p tcp --tcp-flags ALL FIN -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix
"Firewall>FIN scan "
   iptables -A INPUT -p tcp --tcp-flags SYN,RST SYN,RST -m limit --limit 6/h --limit-burst 1 -j LOG --log-
prefix "Firewall>SYN/RST scan "
```

```
   iptables -A INPUT -p tcp --tcp-flags SYN,FIN SYN,FIN -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix
"Firewall>SYN/FIN scan "
   iptables -A INPUT -p tcp --tcp-flags ALL NONE -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix
"Firewall>Null scan "

   # prevent Syn port scanning with psd extension
   if ! locate libxt_psd.so &>/dev/null &&  [ $INTERNET ]
   then
      echo "WARNING: Iptables extension xtables-addons-common must be installed for optimal work."
   fi

   if locate libxt_psd.so &>/dev/null || [ -f /lib/xtables/libxt_psd.so ]
   then
      # Prevent Syn port scan (will detect syn on 5 different port in less than 3 sec)"
      iptables -A INPUT -m psd -p tcp -m limit --limit 6/h --limit-burst 1 -j LOG --log-prefix "Firewall>Syn
scan detected "
   fi

}


# Rules for local traffic
local_traffic ()
{
   iptables -A INPUT -s 127.0.0.0/8 ! -i lo --jump DROP
   iptables -A INPUT -s 127.0.0.1/32 --jump ACCEPT
   iptables -A INPUT -i lo --jump ACCEPT
   iptables -A OUTPUT -o lo --jump ACCEPT
}

# Reset and clean all firewall rules
cleanFirewall ()
{
   # Clean all iptables rules
   iptables -F
   iptables -t nat -F
   iptables -t mangle -Z
   iptables -X
   iptables -t nat -X
   iptables -t mangle -X
   # Restore default policies
   iptables -P INPUT  ACCEPT
   iptables -P OUTPUT  ACCEPT
   iptables -P FORWARD ACCEPT
}

# Mask OS by changing IP stack behaviour
mask_os()
```

```
{
   # Modify default TTL value
   sysctl -w net.ipv4.ip_default_ttl=128 #  Linux normal value is 64
}


# brings nmap -sC into infinit loop (because trying to get infinit header...)
# Also very harmful against other ex Nikto
# Method by Nick Marsh (see http://marc.info/?l=nmap-dev&m=142811327627471&w=2)
http_infinit_tarpit ()
{
   ncat -lkv -p 8080 --sh-exec "echo 'HTTP/1.1 200 OK\r\n'; cat /dev/urandom" # Note: This is not
something to run on producton system, it may eat up your CPU!
}

# Main function
main()
{
   cleanFirewall
   mask_os
   local_traffic
   detect_port_scan
   false_opened_ports
   http_infinit_tarpit
}

main


------------------- End of script --------------------------
```