The Present and Future of Xprobe2
The Next Generation of Active Operating System Fingerprinting

Ofir Arkin
Founder
ofir@sys-security.com
The Sys-Security Group
http://www.sys-security.com

Fyodor Yarochkin
fygrave@tigerteam.net
Beez!LSD Labs
http://www.notlsd.net

Meder Kydyraliev
Meder@areopag.net

July 2003

Abstract

Although some advancement was made in the field of active operating system fingerprinting in the recent years, still, there are many issues to resolve. This paper presents the enhancements made with Xprobe2 v0.2 RC1 and discusses the tool's future development. Both the present and future versions of Xprobe2 introduce many enhancements and advancements to the field of active operating system fingerprinting, which are discussed throughout the paper.

Contents

Figure List

1.0 Introduction

Active operating system fingerprinting is the process of actively determining a targeted network node's underlying operating system by probing the targeted system with several packets and examining the response(s) received.

The traditional approach for active operating system fingerprinting is to examine the TCP/IP stack behavior of a targeted network element when probed with several legitimate and/or malformed packets. The received results would then be compared to a signature database in an effort to find an appropriate match.

Identifying the underlying operating system of a network element, whether remote or local, is an important parameter for the success of many processes in the networking and security arena. Building a network inventory, getting the right context for network intrusion detection systems, and performing a vulnerability analysis are all good examples among many other examples for the use of active operating system fingerprinting.

In the past several years new techniques[1] and advancements[2] were introduced to the field of active operating system fingerprinting. Although these new techniques and mechanisms did enhance the accuracy of active operating system fingerprinting, and significantly lowered the overhead on the network performance by using minimal transmission of packets against a target system to produce the results, the research in the field of active operating system fingerprinting is still incomplete, and the current available open source[3] active operating system fingerprinting tools suffer from several drawbacks.

This paper presents present and future advancements to the field of active operating system fingerprinting which aims to solve different issues with the current used technologies and mechanisms.

[1] Arkin Ofir, "ICMP Usage in Scanning" research project, http://www.sys-security.com
[2] Arkin Ofir & Fyodor Yarochkin, "XProbe2 - A 'Fuzzy' Approach to Remote Active Operating System Fingerprinting", http://www.sys-security.com/html/projects/X.html
[3] In the past several years, commercial tools adopted techniques and enhancements introduced with open source based operating system fingerprinting tools and not vice versa.

2.0 Parameters Effecting the Accuracy of Active Operating System Fingerprinting

Many parameters and issues are effecting the accuracy of the results produced by active operating system fingerprinting tools. Several issues relate to the way these active operating system fingerprinting tools have been engineered, and other issues relate to the topology we perform the scan from, and the topology we perform the scan against.

2.1 The Issue with Hardware-based Devices

The information in a fingerprinting database which relates to an operating system represents the way the operating system (the software) reacts to different fingerprinting tests a tool uses.

With a hardware-based device an entry in the signature database reflects the way the device's firmware (or software) reacts to the different fingerprinting tests.

Fingerprinting according to the hardware device's brand is not doable because of a number of reasons:

- A device is not bound to one firmware version only
- A firmware version might be installed on other devices of the same manufacture

We will either find one firmware version used by all hardware based devices of the same manufacture or a particular firmware version used by a manufacture for a particular functionality (i.e. wireless technology).

A good example is with Cisco based devices using the **I**nternetworking **O**perating **S**ystem (**IOS**). A Cisco 7200 router will be fingerprinted exactly as Cisco's Aironet 1100/1200 wireless access pointes (when both will use IOS version 12.x).

This is one of the reasons why it is impossible to distinguish between different hardware based products, and their functionality, manufactured by Cisco and using IOS, when using traditional active operating system fingerprinting methods. It is possible to identify these devices as manufactured by Cisco and using IOS. It is also possible to divide these devices into groups according to TCP/IP based stack fingerprints differences with the IOS versions they are using, but not to tell what their functionality is[4].

If the designers of a fingerprinting tool failed to understand these issues, the results received, which are based on a corrupted database, will be unreliable.

[4] Even if some other means, such as banner grabbing, will be used against the probed Cisco device, the functionality of the device will not be uncovered.

2.2 The Way Probe Results Are Being Matched

2.2.1 Strict Signature Matching

When results are being received from a targeted system, a strict signature matching based tool would search for a 100% match between the received results and the tool's signature database. If a 100% match is not found, than no match is found and the run fails.

A strict signature matching based tool is extremely sensitive to environmental effects on the probed target, and on the network which the probed target resides on[5].

2.2.2 Signature Matching using Fuzzy Logic

Using fuzzy logic Xprobe2[6] was the first active operating system fingerprinting tool to implement a statistical analysis based mathematical algorithm in order to provide a best effort match between the results received from fingerprinting probes against a targeted system to a signature database. Xprobe2 currently uses one of the simplest forms of **O**ptical **C**haracter **R**ecognition (**OCR**), by utilizing a matrix based fingerprints matching based on statistical calculation of scores for each test performed.

The solution is based on a simple matrix representation of the scan (or scans), and the calculation of 'matches' by simply summing up scores for each 'signature' (Operating System). All tests are performed independently. The following is the abstract matrix used with Xprobe2:

OS **Test**	Operating System *1*	Operating System *2*	Operating System *3*	…	Operating System *i*
Test 1	score	score	score	…	score
Test 2	score		score	…	score
Test 3	score	score	score	…	score
…				…	
Test n	score	score	score		score
Totals	X	Y	Z	…	D

Table 1: The results matrix

Upon initialization each fingerprinting test, which is implemented as an independent module, builds its own vector of possible 'test matches' for each OS (OS $\rightarrow$ (OS_1, OS_2, …OS_i)). This is done by reading the `xprobe2.conf` configuration file, which holds the fingerprints signature database, and searching for the "`fingerprint`" and "`OS_ID`" entries. Once the

[5] Arkin Ofir & Fyodor Yarochkin, "Xprobe2 - A 'Fuzzy' Approach to Remote Active Operating System Fingerprinting", http://www.sys-security.com/archive/papers/Xprobe2.pdf, August 2002.
[6] More information about Xprobe2's way of operation can be found at: http://www.sys-security.com/html/projects/X.html

fingerprinting test is executed the program examines the packet(s) received as the result of the fingerprinting test and places the appropriate '`score`' into the appropriate OS row.

The '`score`' value can take one of the following values:

- YES(3)
- PROBABLY_YES(2)
- PROBABLY_NO(1)
- NO(0).

Each test module assigns the appropriate '`score`' value according to the scheme implemented with the module. Having the '`score`' parameter able to be assigned different values introduces a certain degree of 'fuzziness' with Xprobe2.

Once all tests are completed, a run through is performed through all the columns during which the summary of each operating system is calculated. The top-score OS{x} (X, Y, Z, or D) will be declared as the final result.

This approach gives Xprobe2 probabilistic support since the highest '`score`' given for an operating system (or operating systems) is the most likely to produce an accurate match.

The fuzzy logic implementation with Xprobe2 still misses the ability to assign different weights to different fingerprinting tests. This ability is required since some fingerprinting tests should have bigger impact over the overall fingerprinting results.

Using a fuzzy logic approach to provide a best effort match between the results received from fingerprinting probes against a targeted system to a signature database, provides better resistance against environmental effects which might take their toll on a target system and on probe packets. The quality of the results produced with an active operating system fingerprinting tool utilizing a fuzzy logic approach would be higher. This is if the tool will not suffer from design flaws, and will use a large base of fingerprinting tests.

2.3 The Use of a Fixed Number of Fingerprinting Tests

Active operating system fingerprinting tools usually use a fixed number of tests to determine a remote node's underlying operating system. With each test a fixed number of parameters are being examined. Since the number of tests is fixed, and the number of parameters examined and their possible permutations are also fixed, the total number of possible matches is a fixed number as well.

Although the overall number of possible matches is currently much higher than the number of the current available network elements, certain test classes cannot deliver the expected results and provide with a clear distinction between different network elements.

The conclusion is that a better tool for active operating system fingerprinting would need to utilize fingerprinting tests, which would examine many parameter values in a probe's reply received from a targeted system. These parameter values would need to be different among many network elements. Therefore a number of this kind of tests is required to be used in order to achieve a broader distinction between different network elements. This also suggest that the usage of more parameter rich fingerprinting tests with an active operating fingerprinting tool will provide better overall results.

2.4 The Use of a Certain Fingerprinting Niche

The current available open source active operating system fingerprinting tools usually use a certain niche with the tests they conduct. For example, Xprobe2 v0.1 mainly rely on ICMP based tests where `nmap`[7] mainly rely on TCP based tests. This fixation brings into light the inability of such tools to deal with situations were the fingerprinting tests they use do not yield an adequate result about a certain operating system or even about a class of operating systems.

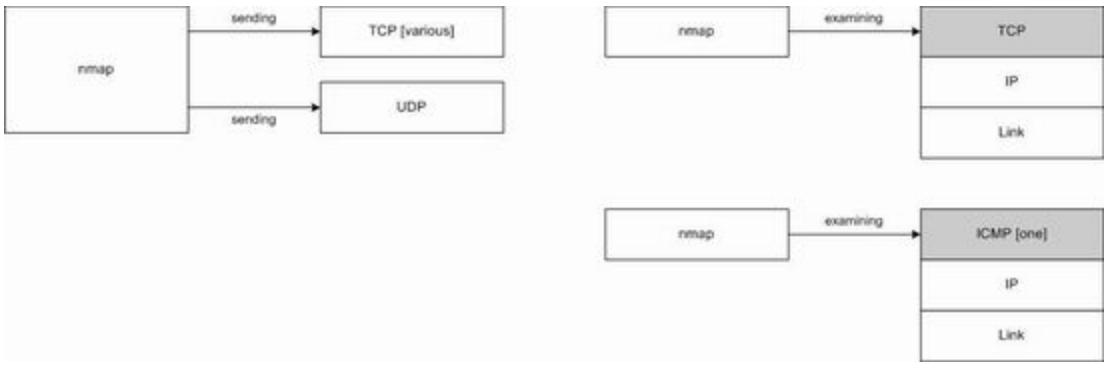

Figure 1: An illustration of `nmap`'s operating system fingerprinting tests

2.5 No Changes are made to the TCP/IP Stacks of New Versions of Operating Systems, or Specific Changes are made With a Certain Protocol's Behavior Only

The behavior of the TCP/IP stack of newly released operating systems hardly changes compared to an older version of the same operating system, or changes made to a newly released operating system's TCP/IP stack effects a certain protocol behavior only. The result would be the inability of some active operating system fingerprinting tools, which rely on a certain fingerprinting niche, to distinguish between different versions of the same operating system or even between a class of the same operating system family.

[7] http://www.insecure.org

With the first example we have initiated an active operating system fingerprinting attempt with Xprobe2 v0.1 against a Sun Solaris 2.6 based machine:

```
[root@angelfire /]# xprobe2 -v x.x.x.x

XProbe2 v.0.1 Copyright (c) 2002-2003 fygrave@tigerteam.net, ofir@sys-security.com,
meder@areopag.net

[+] Target is x.x.x.x
[+] Loading modules.
[+] Following modules are loaded:
        [x][1] ICMP echo (ping)
        [x][2] TTL distance
        [x][3] ICMP echo
        [x][4] ICMP Timestamp
        [x][5] ICMP Address
        [x][6] ICMP Info Request
        [x][7] ICMP port unreach
[+] 7 modules registered
[+] Initializing scan engine
[+] Running scan engine
[+] Host: x.x.x.x is up (Guess probability: 100%)
[+] Target: x.x.x.x is alive
[+] Primary guess:
[+] Host x.x.x.x Running OS: "Sun Solaris 5 (SunOS 2.5)" (Guess probability: 100%)
[+] Other guesses:
[+] Host x.x.x.x Running OS: "Sun Solaris 6 (SunOS 2.6)" (Guess probability: 100%)
[+] Host x.x.x.x Running OS: "Sun Solaris 7 (SunOS 2.7)" (Guess probability: 100%)
[+] Host x.x.x.x Running OS: "Sun Solaris 8 (SunOS 2.8)" (Guess probability: 100%)
[+] Host x.x.x.x Running OS: "Sun Solaris 9 (SunOS 2.9)" (Guess probability: 100%)
[+] Host x.x.x.x Running OS: "Mac OS 9.2.x" (Guess probability: 95%)
[+] Host x.x.x.x Running OS: "HPUX B.11.0 x" (Guess probability: 95%)
[+] Host x.x.x.x Running OS: "Mac OS X 10.1.5" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "FreeBSD 4.3" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "FreeBSD 4.2" (Guess probability: 87%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Using its ICMP based fingerprinting tests, Xprobe2 v0.1 will be unable to distinguish between the Sun Solaris operating systems versions 2.5-2.9.

With the second example we have initiated an active operating system fingerprinting attempt with `nmap` v3.28 against a Microsoft Windows NT 4 based system:

```
[root@angelfire NG]# /usr/local/bin/nmap -sT -O x.x.x.x

Starting nmap 3.28 ( www.insecure.org/nmap/ ) at 2003-06-18 19:14 IDT
Interesting ports on x.x.x.x:
(The 1628 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     filtered    ftp
22/tcp     filtered    ssh
25/tcp     open        smtp
80/tcp     open        http
135/tcp    open        loc-srv
139/tcp    open        netbios-ssn
443/tcp    open        https
465/tcp    open        smtps
1029/tcp   open        ms-lsa
1433/tcp   open        ms-sql-s
2301/tcp   open        compaqdiag
```

```
5555/tcp   open        freeciv
5800/tcp   open        vnc-http
5900/tcp   open        vnc
6000/tcp   filtered    X11
Remote operating system guess: Windows NT 3.51 SP5, NT4 or 95/98/98SE

Nmap run completed -- 1 IP address (1 host up) scanned in 3.334 seconds
```

Using its fingerprinting tests `nmap` is unable to distinguish between different groups of Microsoft based operating systems - XP/2000/ME and NT4/98SE/98/95.

2.6 The Inability to Determine the Exact Software Service Pack Version

Traditional active operating system fingerprinting tools are usually unable to identify the installation of software service packs on a targeted machine. For example, they will identify a target machine is using Microsoft Windows 2000 Server, but will not be able to determine which service pack version is installed (if any at all).

2.7 Some Fingerprinting Tests May Have Bigger Impact on the Overall Results

When using several fingerprinting tests against a targeted system, some fingerprinting tests may have bigger impact on the overall accuracy of the test results compared to other fingerprinting tests.

If these tests fail, for some reason, the quality of the produced results will be lowered significantly, especially with tools using strict signature matching.

The effect of a failure of a mark key test on the results a tool using a fuzzy logic approach produces will be less significant, although it might take its toll as well.

The following are two runs of Xprobe2 against a Microsoft Windows XP based machine. With the first active operating system fingerprinting attempt all fingerprinting modules of Xprobe2 were used, were with the second active operating system fingerprinting attempt we have omitted the ICMP echo fingerprinting test.

```
spanion:~ # xprobe2 -v x.x.x.x

XProbe2 v.0.1 Copyright (c) 2002-2003 fygrave@tigerteam.net, ofir@sys-security.com,
meder@areopag.net

[+] Target is x.x.x.x
[+] Loading modules.
[+] Following modules are loaded:
        [x][1] ICMP echo (ping)
        [x][2] TTL distance
        [x][3] ICMP echo
        [x][4] ICMP Timestamp
        [x][5] ICMP Address
        [x][6] ICMP Info Request
```

```
        [x][7] ICMP port unreach
[+] 7 modules registered
[+] Initializing scan engine
[+] Running scan engine
[+] Host: x.x.x.x is up (Guess probability: 100%)
[+] Target: x.x.x.x is alive
[+] Primary guess:
[+] Host x.x.x.x Running OS: "Microsoft Windows XP Professional / XP Professional SP1"
(Guess probability: 100%)
[+] Other guesses:
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000/2000SP1/2000SP2/2000SP3" (Guess
probability: 100%)
[+] Host x.x.x.x Running OS: "Microsoft Windows 2003 Server" (Guess probability: 95%)
[+] Host x.x.x.x Running OS: "Microsoft Windows ME" (Guess probability: 95%)
[+] Host x.x.x.x Running OS: "Microsoft Windows NT 4 Service Pack 4 and Above" (Guess
probability: 91%)
[+] Host x.x.x.x Running OS: "Microsoft Windows 98/98SE" (Guess probability: 91%)
[+] Host x.x.x.x Running OS: "OpenBSD 2.5" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "NetBSD 1.5.0" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "NetBSD 1.5.1" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "NetBSD 1.5.2" (Guess probability: 87%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

The following is the second active operating system fingerprinting attempt, this time without the ICMP echo fingerprinting module (the usage of the '`-D 3`' option omitted the ICMP echo fingerprinting test):

```
spanion:~ # xprobe2 -v -D 1 -D 2 -D 3 x.x.x.x

XProbe2 v.0.1 Copyright (c) 2002-2003 fygrave@tigerteam.net,
ofir@sys-security.com, meder@areopag.net

[+] Target is x.x.x.x
[+] Loading modules.
[+] Following modules are loaded:
        [x][1] ICMP Timestamp
        [x][2] ICMP Address
        [x][3] ICMP Info Request
        [x][4] ICMP port unreach
[+] 4 modules registered
[+] Initializing scan engine
[+] Running scan engine
[+] All alive tests disabled
[+] Target: x.x.x.x is alive
[+] Primary guess:
[+] Host x.x.x.x Running OS: "Microsoft Windows XP Professional / XP Professional SP1"
(Guess probability: 100%)
[+] Other guesses:
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000/2000SP1/2000SP2/2000SP3" (Guess
probability: 100%)
[+] Host x.x.x.x Running OS: "Microsoft Windows ME" (Guess probability: 100%)
[+] Host x.x.x.x Running OS: "Microsoft Windows 98/98SE" (Guess probability: 94%)
[+] Host x.x.x.x Running OS: "NetBSD 1.5.2" (Guess probability: 94%)
[+] Host x.x.x.x Running OS: "NetBSD 1.5.1" (Guess probability: 94%)
[+] Host x.x.x.x Running OS: "NetBSD 1.5.0" (Guess probability: 94%)
[+] Host x.x.x.x Running OS: "Microsoft Windows NT 4 Service Pack 4 and Above" (Guess
probability: 94%)
[+] Host x.x.x.x Running OS: "NetBSD 1.6.1" (Guess probability: 94%)
[+] Host x.x.x.x Running OS: "OpenBSD 2.5" (Guess probability: 94%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

With the second fingerprinting probe, we have received a perfect match for Microsoft Windows XP, 2000 and ME. Xprobe2 produced nearly the same results even when one of the most important fingerprinting modules, mostly used for identifying Microsoft based operating systems, was not used. The ICMP echo module's extra effect is with distinguishing between a Microsoft Windows 2000/XP operating system and a Microsoft Windows ME operating system. An effect not reflected with the second fingerprinting run which did not include the ICMP echo fingerprinting module.

2.8 Different Networking Devices May Alter A Packet's Field Value

While in transit a packet's content (a field value) might be effected when traversing through different networking devices such as filtering based devices, packet shaping enabled devices and other. These devices may change and/or overwrite a parameter value of one or several fields inside the packet.

We can name several examples:

- A packet shaping device might change several field values within a packet (i.e. a switch enforcing a certain type-of-service (TOS) value)
- A router or a firewall might spoof responses for a targeted system they protect. For example, firewalls, which spoof ICMP query replies for targeted systems they protect, or even perform the TCP 3-way handshake with an initiating system before handing the connection off to a protected targeted system (some sort of a denial-of-service protection[8]).
- A Scrubber[9] may be present between the sending system and the target system.

If a remote active operating system fingerprinting tool relies on a certain IP packet field value, which was altered by the networking environment the packet had traversed, it is more than likely that the matching process would be effected as well and that the results produced would be either inaccurate or false.

[8] For example, Checkpoint Firewall-1

[9] A Scrubber is a software logic aimed at "cleaning" several fields within a packet passing through the machine the Scrubber is installed on. This is done by setting several fields within the packet to certain values so malicious parties trying to determine the remote host's operating system that is located behind the Scrubber, using a remote active operating system fingerprinting technique(s), will fail in their efforts.

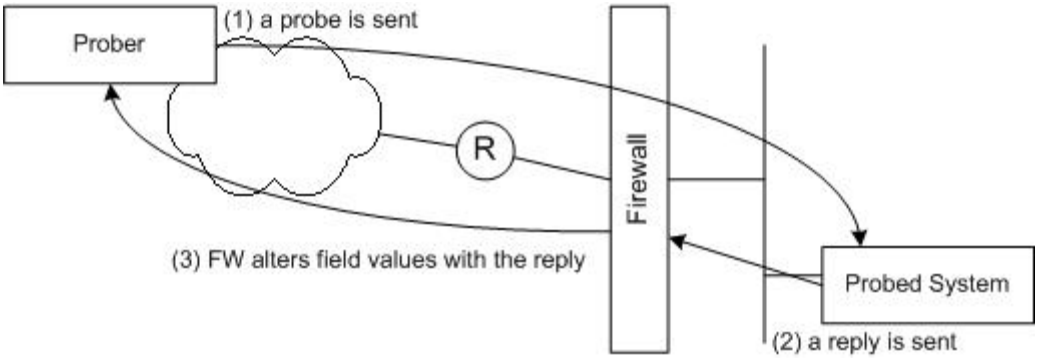

Figure 2: An example for the alteration of packet field value(s) by a firewall

2.9 A Firewalled Target Systems

In a real networking environment some systems will be firewalled. If the traffic filtering is done correctly, then some inbound and outbound packets will be dropped by the firewall (i.e. not allowed in or out). If a remote active operating system fingerprinting tool relies on sending and/or receiving of particular packet types and those packets are dropped by a firewall protecting the target system(s) chances are that the quality of the results will be degraded to the point false results or no results at all will be produced.

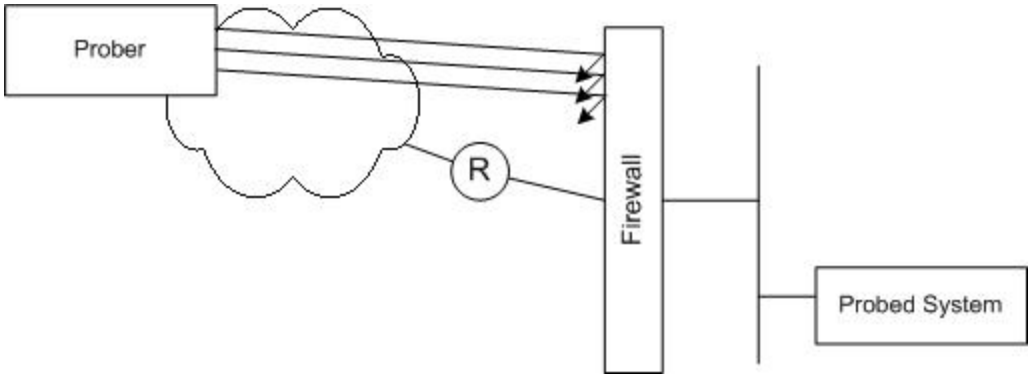

Figure 3: A firewalled target system

2.10 The Use of Malformed Packets

If a remote active operating system fingerprinting tool utilizes malformed packets to produce its fingerprinting results a filtering device may drop these malformed packets, if the filtering device analyzes packets for non-legitimate content[10]. Therefore the quality of the results produced by utilizing a fingerprinting tests relying on malformed packets will be degraded and in some cases even fail.

[10] For example, Checkpoint's Firewall-1 NG with Application Intelligence, http://www.checkpoint.com/appint/index.html.

Malformed packets may have another effect, they might cause some TCP/IP stacks to crash.

2.11 A TCP/IP Stack's Behavior Might Be Altered

Some characteristics of a TCP/IP stack's behavior can be altered by a machine's system administrator:

- Tunable parameters of the TCP/IP stack might be changed e.g. the `sysctl` command on the various *BSDs, the `ndd` command on Sun Solaris[11], etc.
- Numerous patches exist for some open source operating system's kernels that alter the way the particular operating system's TCP/IP stack responses to certain packets[12].

If a remote active operating system fingerprinting tool is using some of the TCP/IP based parameters that can be altered as part of its fingerprinting test, the quality of the results would be effected and questionable, when these parameter values will be altered.

2.12 The Quality of the Signature Database

The quality of the results produced by an active operating system fingerprinting tool is not only a factor of programming and terrain. It is much effected from the way the signature database of the tool was and is built. If signatures submitted to the database were and are obtained in a wrongfully manner than the signature database should be regarded as corrupt. The results produced by the tool will not be accurate.

A signature should be built in a lab environment where the optimal conditions are to produce an accurate signature are maximal.

2.13 The Inability to Identify the Underlying Architecture Platform

Usually, active operating system fingerprinting tools will identify the operating system of a network node, but not its underlying platform.

The knowledge about the underlying platform is extremely important for tools performing vulnerability assessment, network inventory, etc. which rely on the results of the active operating system fingerprinting tool (i.e. nessus).

---

[11] For a complete list of Sun Solaris 9 tunable parameters please see: "Solaris Tunable Parameters Reference Manual", http://docs.sun.com/db/doc/806-7009.

[12] One example is the IP Personality patch for Linux Kernel 2.4.x, http://ippersonality.sourceforge.net/.

2.14 The Inability to Scale

Another important parameter for an active operating system fingerprinting tool is the ability to scan large networks such as class B networks. Here comes into play the number of packets needed for an active operating system fingerprinting tool in order to determine the remote machine's underlying operating system.

For example, `nmap` would use 98,302,500 (1500 packets on average per host) packets in order to scan and determine the operating system of network nodes available on a class B network (assuming all 65535 addresses are assigned).

When scanning a large-scale network an important parameter must be taken into considered – for any router and switch there is a limit to the number of packets per second it can process. Beyond that limit, some packets will be dropped, but more important, the router/switch might suffer from a denial of service condition. Therefore it is very important to balance the scan rate with the network and network elements abilities.

On the other hand an active operating system fingerprinting tool must not be too slow to scan large-scale networks, and it needs to do that effectively.

2.15 Inability to Control the Fingerprinting Modules to Be Executed

The motive for performing an active operating system fingerprinting attempt is an important parameter that needs to be taken into account. Usually for each motive a different scanning tactic or sets of tactics is to be used.

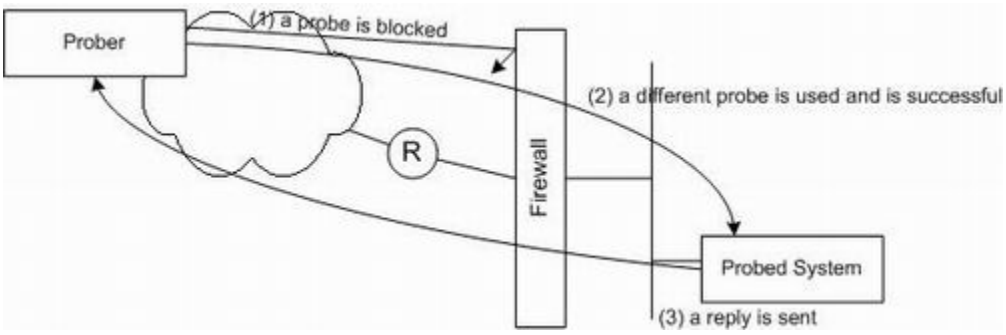

Figure 4: Changing scanning tactics

If the intent of the scan is malicious, than it might be more beneficial for the malicious attacker to use minimum packets to achieve maximum results.

If the reason behind performing the scan is not malicious, usually, there is less importance for being stealth, still needing to achieve maximum results. Although we would still like to send minimum packets to probe each targeted system, eliminating the possibility of overwhelming the network and effecting its performance, we can use whatever fingerprinting tests we have for our disposal.

Therefore we would like to have the ability to control which tests a particular active operating system fingerprinting tool uses against a targeted system. Furthermore, we would like an active operating system fingerprinting tool to be able to detect certain scanning conditions and to react, by switching scanning tactics (i.e. scanning a host behind a firewall).

When scanning different machines on different topologies some tests would be proved useless. Controlling which tests to use would result with better accuracy and less chance of being detected.

3.0 The Present - Xprobe2 v0.2 RC1 Enhanced Functionality

The functionality and capabilities of Xprobe2 were considerably enhanced with the introduction of Xprobe2 v0.2 RC1. These enhancements are designed to improve several operational aspects, as well as to allow one complete control over the tool's operation. The enhancements allow producing better results against different topologies, with several parameters getting automatic adjustments according to the terrain.

The fact that Xprobe2 uses fuzzy logic in order to match the results of an active operating system fingerprinting attempt against a targeted system with Xprobe2's signature database, allows Xprobe2 to overcome many issues and obstacles which other active operating system fingerprinting tools currently face.

3.1 New Discovery Modules

Xprobe2's discovery modules are designed to perform host detection, firewall detection, and to provide information for the automatic receive timeout mechanism.

With Xprobe2 v0.2 RC1 two new discovery modules were introduced, the "TCP ping" and "UDP ping" discovery modules.

The "TCP ping" and "UDP ping" discovery modules are not executed by default. One must specify, using the "-p" command line option, an open or a closed TCP port for the "TCP ping" discovery module to be executed, and a closed UDP port for the "UDP ping" discovery module to be executed.

The aim of the discovery modules is to elicit a response from a targeted host, either a SYN|ACK or a RST as a response for the "TCP ping" discovery module and an ICMP port unreachable as a response for the "UDP ping" discovery module. The round trip time calculated for a successful run of a discovery module is used with the automatic receives timeout mechanism (more on that later).

3.2 The Ability to Control Xprobe2's Modules Execution

The ability to control Xprobe2's modules execution was greatly enhanced with the introduction of the "`-D`", and "`-M`" command line options. They allow one to control which Xprobe2 modules to be executed:

- With the "-D" command line option one can specify which Xprobe2 modules *not* to use
- With the "-M" command line option one can specify which Xprobe2 modules to use

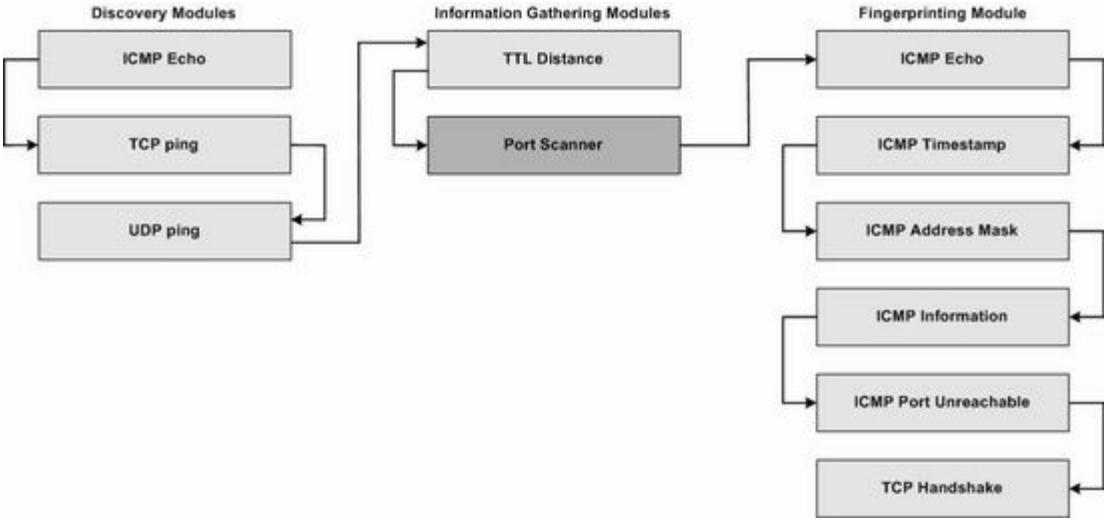

Figure 5: Xprobe2 modules and their execution order

The following is an example of using the "-D" command line option. With the example below we have performed a remote operating system fingerprinting attempt against a Microsoft Windows 2000 Server with service pack 3. The probe did not use two modules, the TTL distance reachability module, and the information request fingerprinting module. Please note that both "-D" and "-M" command line options can use numerical and/or a module name as parameters:

```
[root@fremont src]# ./xprobe2 -v -c ../etc/xprobe2.conf -D infogather:ttl_calc -D 9 -p
TCP:139:open x.x.x.x

Xprobe2 v.0.2 Copyright (c) 2002-2003 fygrave@tigerteam.net, ofir@sys-security.com,
meder@areopag.net

[+] Target is x.x.x.x
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:portscan  -  TCP and UDP PortScanner
[x] [5] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [6] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [7] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [8] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[x] [9] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[+] 9 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:udp_ping module: no closed/open UDP ports known on x.x.x.x. Module test failed
[+] Host: x.x.x.x is up (Guess probability: 66%)
[+] Target: x.x.x.x is alive. Round-Trip Time: 0.09818 sec
[+] Selected safe Round-Trip Time value is: 0.19636 sec
[+] Primary guess:
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000 Server Service Pack 3" (Guess
probability: 100%)
[+] Other guesses:
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000 Workstation SP1" (Guess probability:
96%)
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000 Workstation SP2" (Guess probability:
```

```
96%)
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000 Workstation SP3" (Guess probability:
96%)
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000 Workstation SP4" (Guess probability:
96%)
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000 Server" (Guess probability: 96%)
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000 Server Service Pack 1" (Guess
probability: 96%)
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000 Server Service Pack 2" (Guess
probability: 96%)
[+] Host x.x.x.x Running OS: "Microsoft Windows 2000 Server Service Pack 4" (Guess
probability: 96%)
[+] Host x.x.x.x Running OS: "Microsoft Windows XP" (Guess probability: 96%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

The following is an example for the usage of the "-M" command line option. With the example below we have performed a remote operating system fingerprinting attempt against a Cisco router using IOS version 11.1. The probe used three modules, the ICMP echo reachability test, the ICMP echo fingerprinting test, and the ICMP port unreachable fingerprinting test:

```
[root@fremont src]# ./xprobe2 -v -c ../etc/xprobe2.conf -M ping:icmp_ping -M
fingerprint:icmp_echo -M fingerprint:icmp_port_unreach -p TCP:23:open x.x.x.x

Xprobe2 v.0.2 Copyright (c) 2002-2003 fygrave@tigerteam.net,
ofir@sys-security.com, meder@areopag.net

[+] Target is x.x.x.x
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [3] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[+] 3 modules registered
[+] Initializing scan engine
[+] Running scan engine
[+] Host: x.x.x.x is up (Guess probability: 100%)
[+] Target: x.x.x.x is alive. Round-Trip Time: 0.05988 sec
[+] Selected safe Round-Trip Time value is: 0.11975 sec
[+] Primary guess:
[+] Host x.x.x.x Running OS: "Cisco IOS 11.2" (Guess probability: 100%)
[+] Other guesses:
[+] Host x.x.x.x Running OS: "Cisco IOS 11.1" (Guess probability: 100%)
[+] Host x.x.x.x Running OS: "NetBSD 1.5.3" (Guess probability: 93%)
[+] Host x.x.x.x Running OS: "NetBSD 1.5.2" (Guess probability: 93%)
[+] Host x.x.x.x Running OS: "NetBSD 1.5.1" (Guess probability: 93%)
[+] Host x.x.x.x Running OS: "NetBSD 1.5" (Guess probability: 93%)
[+] Host x.x.x.x Running OS: "OpenBSD 2.5" (Guess probability: 93%)
[+] Host x.x.x.x Running OS: "NetBSD 1.4.3" (Guess probability: 93%)
[+] Host x.x.x.x Running OS: "NetBSD 1.4.2" (Guess probability: 93%)
[+] Host x.x.x.x Running OS: "Cisco IOS 11.3" (Guess probability: 93%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

The "-L" command line option can be used to list available modules.

Except for the "TCP ping", "UDP ping", and the port scanner module, all Xprobe2's modules will be executed by default.

Conclusion
Combined with Xprobe2's other command line options (such as "-s" and "-t"), the complete control over the tool's operations and usage is given to the end user.

This complete control over Xprobe2's way of operation allows one to execute different modules according to the topology it is facing.

3.3 The Usage of Best of Breed Active Operating System Fingerprinting Technologies and Techniques

An active operating system fingerprinting tool must use best of breed TCP/IP stack based operating system fingerprinting techniques and methods. Focusing on one niche only is not useful in the long run. Therefore we have added a TCP-based operating system fingerprinting module to Xprobe2.

3.3.1 The TCP Handshake Module

We have decided on adding a TCP-based operating system fingerprinting module which will provide with a strong impact over the overall fingerprinting results. A fingerprinting test which will use as much parameters as possible and will provide with a real added value. We have decided on adding a TCP fingerprinting module based on the TCP 3-way handshake process.

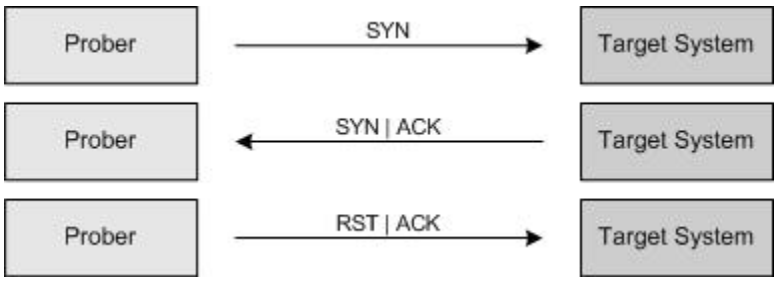

Figure 6: The TCP 3-way handshake

With Xprobe2's TCP handshake module a SYN request is sent to an open TCP port on a targeted system. The parameters of the SYN request resemble the parameters used with the Linux operating system's `telnet` request. Parameters with the received TCP SYN | ACK response are then examined as part of the fingerprinting process. A RST will be sent to the targeted system to tear down the connection.

The parameters examined with Xprobe2's TCP handshake module are the:

- IP header's type of service field value
- IP header's don't fragment bit

- IP header's IP ID field value
- IP header's time-to-live field value
- TCP acknowledgment number
- TCP initial windows size
- TCP options and their order of appearance
- TCP options window scale field value

Unlike other tools, which use a similar module, Xprobe2 examines parameters found in the IP and TCP layers.

The following is an example of the impact the TCP handshake module has on the overall results Xprobe2 produces. The following fingerprinting attempts were produced against a Sun Solaris 2.8 based machine. The first run did not include the TCP handshake module ("-D 11"):

```
[root@fremont src]# ./xprobe2 -v -c ../etc/xprobe2.conf -p TCP:25:open -D 11 x.x.x.x

Xprobe2 v.0.2 Copyright (c) 2002-2003 fygrave@tigerteam.net,
ofir@sys-security.com, meder@areopag.net

[+] Target is x.x.x.x
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_info  -  ICMP Information request fingerprinting module
[x] [10] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[+] 10 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:udp_ping module: no closed/open UDP ports known on x.x.x.x. Module test failed
[+] Host: x.x.x.x is up (Guess probability: 75%)
[+] Target: x.x.x.x is alive. Round-Trip Time: 0.00156 sec
[+] Selected safe Round-Trip Time value is: 0.00312 sec
[+] Primary guess:
[+] Host x.x.x.x Running OS: "Sun Solaris 6 (SunOS 2.6)" (Guess probability: 100%)
[+] Other guesses:
[+] Host x.x.x.x Running OS: "Sun Solaris 7 (SunOS 2.7)" (Guess probability: 100%)
[+] Host x.x.x.x Running OS: "Sun Solaris 8 (SunOS 2.8)" (Guess probability: 100%)
[+] Host x.x.x.x Running OS: "Sun Solaris 9 (SunOS 2.9)" (Guess probability: 100%)
[+] Host x.x.x.x Running OS: "HP UX 11.0" (Guess probability: 95%)
[+] Host x.x.x.x Running OS: "HP UX 11.0i" (Guess probability: 91%)
[+] Host x.x.x.x Running OS: "OpenBSD 2.5" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "NetBSD 1.4" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "NetBSD 1.4.1" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "NetBSD 1.4.2" (Guess probability: 87%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

The second run included the TCP handshake module:

```
[root@fremont src]# ./xprobe2 -v -c ../etc/xprobe2.conf -p TCP:25:open x.x.x.x

Xprobe2 v.0.2 Copyright (c) 2002-2003 fygrave@tigerteam.net,
ofir@sys-security.com, meder@areopag.net

[+] Target is x.x.x.x
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_info  -  ICMP Information request fingerprinting module
[x] [10] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[x] [11] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[+] 11 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:udp_ping module: no closed/open UDP ports known on x.x.x.x. Module test failed
[+] Host: x.x.x.x is up (Guess probability: 75%)
[+] Target: x.x.x.x is alive. Round-Trip Time: 0.09717 sec
[+] Selected safe Round-Trip Time value is: 0.19434 sec
[+] Primary guess:
[+] Host x.x.x.x Running OS: "Sun Solaris 8 (SunOS 2.8)" (Guess probability: 100%)
[+] Other guesses:
[+] Host x.x.x.x Running OS: "Sun Solaris 7 (SunOS 2.7)" (Guess probability: 93%)
[+] Host x.x.x.x Running OS: "Sun Solaris 9 (SunOS 2.9)" (Guess probability: 93%)
[+] Host x.x.x.x Running OS: "Sun Solaris 6 (SunOS 2.6)" (Guess probability: 90%)
[+] Host x.x.x.x Running OS: "HP UX 11.0" (Guess probability: 90%)
[+] Host x.x.x.x Running OS: "HP UX 11.0i" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "OpenBSD 2.5" (Guess probability: 81%)
[+] Host x.x.x.x Running OS: "OpenBSD 2.9" (Guess probability: 81%)
[+] Host x.x.x.x Running OS: "NetBSD 1.4" (Guess probability: 81%)
[+] Host x.x.x.x Running OS: "NetBSD 1.4.1" (Guess probability: 81%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Combined with Xprobe2's other fingerprinting modules, the TCP handshake module greatly enhance Xprobe2's abilities, overall accuracy, and the ability to provide results when executed against different topologies.

3.4 Port Scanner

The success of executing some of Xprobe2's fingerprinting modules depends on successfully probing an open TCP port and a closed UDP port. Therefore we have implemented a port scanner module as an independent module to Xprobe2. The port scanner module allows one to discover open and closed UDP and TCP ports, and later use the discoveries as parameters for the operating system fingerprinting attempt. By default Xprobe2 will not execute the port scanner module and will not tie the module with its fingerprinting modules. Therefore Xprobe2 still maintains its minimal usage of packets to discover a targeted system's underlying operating system.

In order to use Xprobe2's port scanner module one needs to use the '-P' command line option combined with either the '-T' command line option, for TCP-based port scanning, and/or the '-U' command line option, for UDP-based port scanning, and to specify the port/ports to be probed.

The following example is an operating system fingerprinting attempt produced against a targeted HP-UX 11.0 based machine. The port scanner is used to scan TCP ports 20 to 40 and 80:

```
[root@fremont src]# ./xprobe2 -v -c ../etc/xprobe2.conf -s 0.1 -P -T 20-40,80 x.x.x.x

Xprobe2 v.0.2 Copyright (c) 2002-2003 fygrave@tigerteam.net,
ofir@sys-security.com, meder@areopag.net

[+] Target is x.x.x.x
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_info  -  ICMP Information request fingerprinting module
[x] [10] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[x] [11] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[+] 11 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on x.x.x.x. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on x.x.x.x. Module test failed
[+] No distance calculation. x.x.x.x appears to be dead or no ports known
[+] Host: x.x.x.x is up (Guess probability: 25%)
[+] Target: x.x.x.x is alive. Round-Trip Time: 0.00149 sec
[+] Selected safe Round-Trip Time value is: 0.00298 sec
[+] Portscan results for x.x.x.x:
[+]  Stats:
[+]   TCP: 4 - open, 18 - closed, 0 - filtered
[+]   UDP: 0 - open, 0 - closed, 0 - filtered
[+]   Portscan took 2.50 seconds.
[+]  Details:
[+]   Proto     Port Num.       State           Serv. Name
[+]   TCP       21              open            ftp
[+]   TCP       22              open            ssh
[+]   TCP       23              open            telnet
[+]   TCP       37              open            time
[+]  Other ports are in closed state.
[+] Primary guess:
[+] Host x.x.x.x Running OS: "HP UX 11.0" (Guess probability: 100%)
[+] Other guesses:
[+] Host x.x.x.x Running OS: "HP UX 11.0i" (Guess probability: 96%)
[+] Host x.x.x.x Running OS: "Sun Solaris 8 (SunOS 2.8)" (Guess probability: 90%)
[+] Host x.x.x.x Running OS: "Sun Solaris 9 (SunOS 2.9)" (Guess probability: 90%)
[+] Host x.x.x.x Running OS: "Sun Solaris 6 (SunOS 2.6)" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "Sun Solaris 7 (SunOS 2.7)" (Guess probability: 87%)
[+] Host x.x.x.x Running OS: "OpenBSD 2.5" (Guess probability: 78%)
[+] Host x.x.x.x Running OS: "OpenBSD 2.9" (Guess probability: 78%)
[+] Host x.x.x.x Running OS: "NetBSD 1.4" (Guess probability: 78%)
[+] Host x.x.x.x Running OS: "NetBSD 1.4.1" (Guess probability: 78%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

When the port scanner module is used, knowledge about opened TCP ports, and closed UDP ports will be used as parameters for other modules. With the example above, since the "-p" command line option was not used, the port used for the TCP handshake fingerprinting module was one that was already discovered as an opened TCP port by the port scanner. The TCP handshake module used TCP port 21 to perform its fingerprinting. A port that was discovered as opened by the port scanner:

```
06:43:11.816561 y.y.y.y.58662 > x.x.x.x.21: S [tcp sum ok] 3680439203:3680439203(0) win
5840 [tos 0x10]  (ttl 64, id 36350, len 40)
                        4510 0028 8dfe 0000 4006 b52d yyyy yyyy
                        xxxx xxxx e526 0015 db5f 0ba3 0000 0000
                        5002 16d0 953f 0000
06:43:11.816905 x.x.x.x.21 > y.y.y.y.58662: S [tcp sum ok] 1267438502:1267438502(0) ack
3680439204 win 32768 <mss 536> (DF) (ttl 64, id 38918, len 44)
                        4500 002c 9806 4000 4006 6b31 xxxx xxxx
                        yyyy yyyy 0015 e526 4b8b 93a6 db5f 0ba4
                        6012 8000 38ac 0000 0204 0218 0000
06:43:11.817129 y.y.y.y.58662 > x.x.x.x.21: R [tcp sum ok] 3680439204:3680439204(0) win 0
(DF) [tos 0x10]  (ttl 64, id 0, len 40)
                        4510 0028 0000 4000 4006 032c yyyy yyyy
                        xxxx xxxx e526 0015 db5f 0ba4 0000 0000
                        5004 0000 ac0c 0000
06:43:11.884641 y.y.y.y.9528 > x.x.x.x.22: S [tcp sum ok] 3430926786:3430926786(0) win
5840 [tos 0x10]  (ttl 64, id 21348, len 40)
                        4510 0028 5364 0000 4006 efc7 yyyy yyyy
                        xxxx xxxx 2538 0016 cc7f c9c2 0000 0000
                        5002 16d0 a5ed 0000
06:43:11.884973 x.x.x.x.22 > y.y.y.y.9528: R [tcp sum ok]
0:11(11) ack 3430926787 win 0 [RST No listener] (DF) [tos 0x10]  (ttl 64, id 38919, len
51)
                        4510 0033 9807 4000 4006 6b19 xxxx xxxx
                        yyyy yyyy 0016 2538 0000 0000 cc7f c9c3
                        5014 0000 8f85 0000 4e6f 206c 6973 7465
                        6e65 72
06:43:11.916201 y.y.y.y.1937 > x.x.x.x.23: S [tcp sum ok] 3025082354:3025082354(0) win
5840 (ttl 64, id 42256, len 40)
                        4500 0028 a510 0000 4006 9e2b yyyy yyyy
                        xxxx xxxx 0791 0017 b44f 17f2 0000 0000
                        5002 16d0 8d94 0000
06:43:11.916507 x.x.x.x.23 > y.y.y.y.1937: S [tcp sum ok] 1267596579:1267596579(0) ack
3025082355 win 32768 <mss 536> (DF) (ttl 64, id 38920, len 44)
                        4500 002c 9808 4000 4006 6b2f xxxx xxxx
                        yyyy yyyy 0017 0791 4b8d fd23 b44f 17f3
                        6012 8000 c781 0000 0204 0218 0000
06:43:11.916574 y.y.y.y.1937 > x.x.x.x.23: R [tcp sum ok] 3025082355:3025082355(0) win 0
(DF) (ttl 64, id 0, len 40)
                        4500 0028 0000 4000 4006 033c yyyy yyyy
                        xxxx xxxx 0791 0017 b44f 17f3 0000 0000
                        5004 0000 a461 0000
...
```

The Handshake module execution:

```
06:43:12.567945 y.y.y.y.9147 > x.x.x.x.21: S [tcp sum ok] 1226003832:1226003832(0) win
5840 <mss 1460,sackOK,timestamp 567759 0,nop,wscale 0> (DF) [tos 0x10]  (ttl 64, id
47907, len 60)
                        4510 003c bb23 4000 4006 47f4 yyyy yyyy
                        xxxx xxxx 23bb 0015 4913 5578 bb23 270e
                        a002 16d0 ab3d 0000 0204 05b4 0402 080a
                        0008 a9cf 0000 0000 0103 0300
06:43:12.568220 x.x.x.x.21 > y.y.y.y.9147: S [tcp sum ok] 1267978727:1267978727(0) ack
```

```
1226003833 win 32768 <mss 1460,nop,nop,sackOK,wscale 0,nop,nop,nop,timestamp 1280620614
567759> (DF) (ttl 64, id 38946, len 64)
                        4500 0040 9822 4000 4006 6b01 xxxx xxxx
                        yyyy yyyy 0015 23bb 4b93 d1e7 4913 5579
                        b012 8000 f110 0000 0204 05b4 0101 0402
                        0303 0001 0101 080a 4c54 b846 0008 a9cf
06:43:12.568278 y.y.y.y.9147 > x.x.x.x.21: R [tcp sum ok] 1226003833:1226003833(0) win 0
(DF) [tos 0x10]  (ttl 64, id 0, len 40)
                        4510 0028 0000 4000 4006 032c yyyy yyyy
                        xxxx xxxx 23bb 0015 4913 5579 0000 0000
                        5004 0000 b5ef 0000
```

The following example is an operating system fingerprinting attempt against a targeted Linux Kernel 2.2.5 based machine[13]. The port scanner is used to scan TCP ports 20 to 30 and UDP ports 516 to 520:

```
spanion:~/tmp/xprobe2-0.2rc1/src # ./xprobe2 -v -c ../etc/xprobe2.conf -s 0.05 -P -U 516-
520 -T 20-30 192.168.0.150

Xprobe2 v.0.2rc1 Copyright (c) 2002-2003 fygrave@tigerteam.net, ofir@sys-security.com,
meder@areopag.net

[+] Target is 192.168.0.150
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_info  -  ICMP Information request fingerprinting module
[x] [10] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[x] [11] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[+] 11 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 192.168.0.150. Module test
failed
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.0.150. Module test
failed
[+] No distance calculation. 192.168.0.150 appears to be dead or no ports known
[+] Host: 192.168.0.150 is up (Guess probability: 25%)
[+] Target: 192.168.0.150 is alive. Round-Trip Time: 0.00128 sec
[+] Selected safe Round-Trip Time value is: 0.00257 sec
[+] Portscan results for 192.168.0.150:
[+]  Stats:
[+]   TCP: 3 - open, 8 - closed, 0 - filtered
[+]   UDP: 2 - open, 3 - closed, 0 - filtered
[+]   Portscan took 0.97 seconds.
[+]  Details:
[+]   Proto     Port Num.       State           Serv. Name
[+]   TCP       21              open            ftp
[+]   TCP       23              open            telnet
[+]   TCP       25              open            smtp
[+]   UDP       517             open            talk
[+]   UDP       518             open            ntalk
[+]  Other ports are in closed state.
[+] Primary guess:
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.22" (Guess probability: 100%)
[+] Other guesses:
```

[13] All Linux Kernel 2.2.x based machines are fingerprinted the same.

```
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.23" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.24" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.25" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.8" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.7" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.6" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.5" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.4" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.3" (Guess probability: 100%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

The following diagram describes the relationship between the usage and discoveries of the port scanner module and the "-p" command line option:

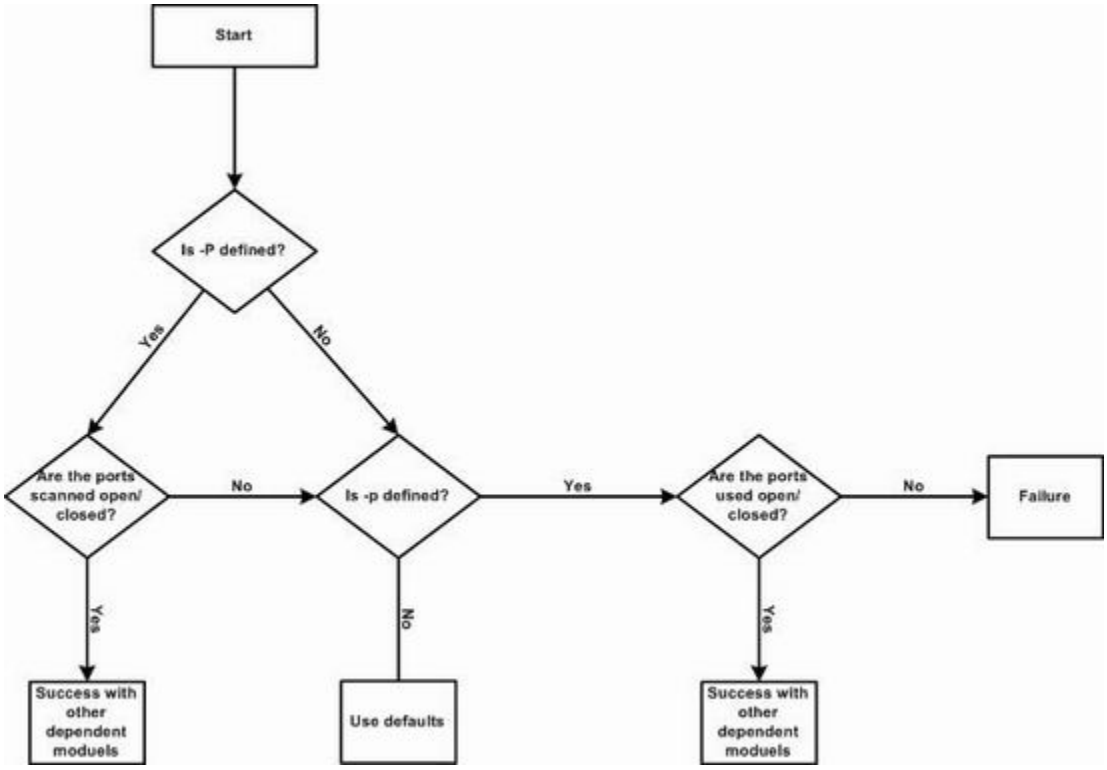

Figure 7: The relationship between "-P" and "-p" command line options with Xprobe 0.2 RC1

Currently with Xprobe2 v0.2RC1 the modules which receive input from the port scanner module are the ICMP port unreachable fingerprinting module (closed UDP port) and the TCP handshake fingerprinting module (opened TCP port).

3.4.1 Controlling the Sending Stream

A command line option, "-s", was added to control the sending stream of packets done by the port scanner module. The command line controls the time interval between each SYN packet sent and/or UDP datagram sent. The value given is represented in milliseconds.

If the "-s" command line option is not used, Xprobe2 RC1 will use a default time interval of 10 milliseconds between SYN packets sent and/or UDP datagram sent (i.e. 100 packets per second).

Controlling the stream of packets the port scanner will generate is an important feature, allowing one to adjust the paste of the scan, not allowing denial of service conditions to be introduced (against networking gear the packets go through, or even against the targeted machine), and adjusting the port scanner's paste to accommodate network and host related issues (the network is congested, old networking gear, etc.).

In some situations one must use the "-s" option to specify a longer delay between packets sent, since a target operating system (i.e. FreeBSD, Cisco routers) might have an automatic feature to rate-limit the number of replies it sends per a certain amount of time.

The following example illustrates the importance of using the "-s" command line option. Since the command line option effects the receive timeout calculation for the port scanner module (more on that later) it has an important effect over the port scanning results. With the first run the "-s" command line option was not used. Please note that with the results of the port scanner one of the TCP ports is listed as "filtered":

```
spanion:~/tmp/xprobe2-0.2rc1/src # ./xprobe2 -v -c ../etc/xprobe2.conf -P -U 516-520 -T
20-30  192.168.0.150

Xprobe2 v.0.2rc1 Copyright (c) 2002-2003 fygrave@tigerteam.net, ofir@sys-security.com,
meder@areopag.net

[+] Target is 192.168.0.150
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_info  -  ICMP Information request fingerprinting module
[x] [10] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[x] [11] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[+] 11 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 192.168.0.150. Module test
failed
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.0.150. Module test
failed
[+] No distance calculation. 192.168.0.150 appears to be dead or no ports known
[+] Host: 192.168.0.150 is up (Guess probability: 25%)
[+] Target: 192.168.0.150 is alive. Round-Trip Time: 0.00064 sec
[+] Selected safe Round-Trip Time value is: 0.00128 sec
```

```
[+] Portscan results for 192.168.0.150:
[+]  Stats:
[+]   TCP: 3 - open, 7 - closed, 1 - filtered
[+]   UDP: 2 - open, 3 - closed, 0 - filtered
[+]   Portscan took 0.34 seconds.
[+]  Details:
[+]   Proto     Port Num.       State           Serv. Name
[+]   TCP       21              open            ftp
[+]   TCP       23              open            telnet
[+]   TCP       25              open            smtp
[+]   TCP       30              filtered        N/A
[+]   UDP       517             open            talk
[+]   UDP       518             open            ntalk
[+]  Other ports are in closed state.
[+] Primary guess:
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.22" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.23" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.24" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.25" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.8" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.7" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.6" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.5" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.4" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.3" (Guess probability: 100%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

With the second run we have used the "-s" command line option to specify the sending time interval and to effect the receive timeout of the port scanner module (a longer timeout). With the second run answers were received from all probed ports (no "filtered" ports):

```
spanion:~/tmp/xprobe2-0.2rc1/src # ./xprobe2 -v -c ../etc/xprobe2.conf -s 0.04 -P -U 516-
520 -T 20-30  192.168.0.150

Xprobe2 v.0.2rc1 Copyright (c) 2002-2003 fygrave@tigerteam.net, ofir@sys-security.com,
meder@areopag.net

[+] Target is 192.168.0.150
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_info  -  ICMP Information request fingerprinting module
[x] [10] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[x] [11] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[+] 11 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 192.168.0.150. Module test
failed
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.0.150. Module test
failed
[+] No distance calculation. 192.168.0.150 appears to be dead or no ports known
[+] Host: 192.168.0.150 is up (Guess probability: 25%)
[+] Target: 192.168.0.150 is alive. Round-Trip Time: 0.00080 sec
[+] Selected safe Round-Trip Time value is: 0.00159 sec
[+] Portscan results for 192.168.0.150:
```

```
[+]  Stats:
[+]   TCP: 3 - open, 8 - closed, 0 - filtered
[+]   UDP: 2 - open, 3 - closed, 0 - filtered
[+]   Portscan took 0.81 seconds.
[+]  Details:
[+]   Proto     Port Num.       State           Serv. Name
[+]   TCP       21              open            ftp
[+]   TCP       23              open            telnet
[+]   TCP       25              open            smtp
[+]   UDP       517             open            talk
[+]   UDP       518             open            ntalk
[+]  Other ports are in closed state.
[+] Primary guess:
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.22" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.23" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.24" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.25" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.8" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.7" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.6" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.5" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.4" (Guess probability: 100%)
[+] Host 192.168.0.150 Running OS: "Linux Kernel 2.2.3" (Guess probability: 100%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

3.5 A Mechanism for Automatic Calculation of the Receive Timeout

An automatic receive timeout calculation mechanism was implemented with Xprobe2 v0.2 RC1. The mechanism allows Xprobe2 to be time efficient taking into account the topology it works in and against.

Xprobe2 uses three receive timeouts:

- For its discovery modules (ICMP echo (ping), "TCP ping", and "UDP ping")
- For the port scanner module, and
- For the fingerprinting modules

Xprobe2 uses the three different discovery modules in order to calculate the receiving timeout for its fingerprinting modules. The timeout used is the longest round-trip time of a discovery modules used in a probe times two (RTT*2). In order to allow a proper receive timeout for the discovery modules themselves, one can use the "–t" command line option and specify the receiving timeout in milliseconds.

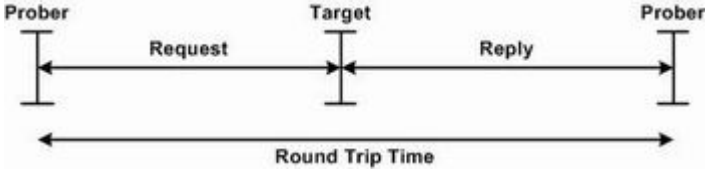

Figure 8: Round Trip Time

The port scanner's receive timeout is calculated differently:

(The number of ports to scan * ((sending delay defined by "-s") + 0.01)) + RTT*2)

The number of ports to be scanned is multiplied by the sending delay, specified by the "-s" command line option, plus a constant number (used in case "-s" is not defined). To that number we add the RTT*2 which was calculated earlier using the discovery modules. The final result, in milliseconds, is the receiving timeout for the port scanner module.

If the number of received replies equals the number of probes sent, the port scanner module will time out before the receiving timeout has been reached.

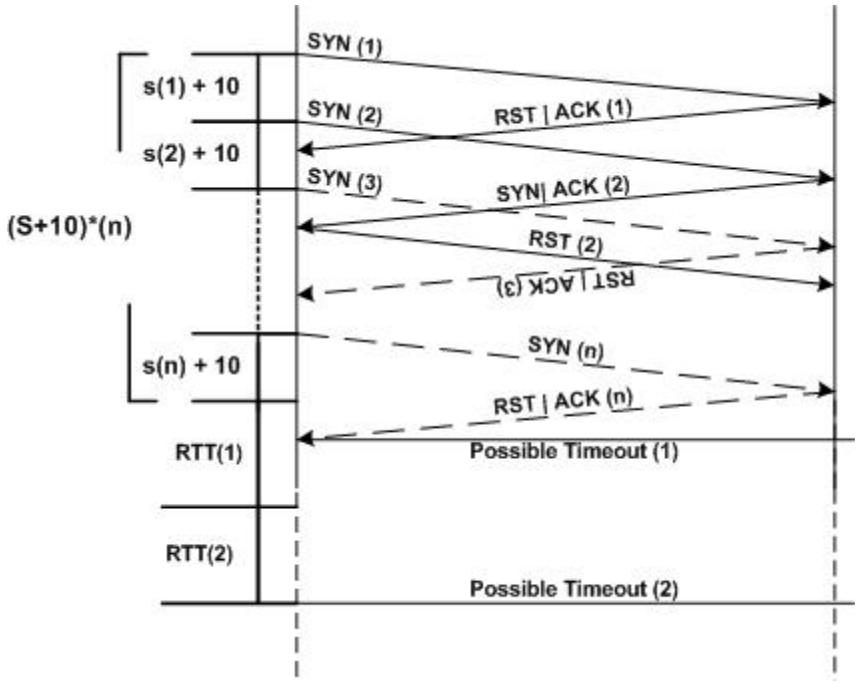

Figure 9: The Receive Timeout of the Port Scanner module

3.6 Maintaining a Quality Signature Database

Xprobe2's signature database is tightly controlled. New signatures will be added to the database if, and only if, we can verify them against a test system we control or have legitimate access to.

We see the signature database issue as a mandatory issue for the success of Xprobe2. It is very easy to corrupt a signature database where it would lead to false and inaccurate results.

Tightly controlling the signature database suggests that the signature database of Xprobe2 is smaller than other signature databases of other active operating system fingerprinting tools. Although it might be smaller in size than other signature databases, the accuracy of Xprobe2's signature database cannot be matched by any other active operating system fingerprinting tool. Xprobe2 maintains the ability to easily add new fingerprinting modules.

The signature database of Xprobe2 v0.2 RC1 was rebuilt from scratch. Each signature was verified against the device or operating system it represents.

4.0 The Future of Active Operating System Fingerprinting

The *terrain* we scan from and against is a major factor in producing accurate results with active operating system fingerprinting tools. One cannot compare the *scanning conditions* of a local network versus a well fortified Internet web site. With the latter the number of TCP/IP based stack operating system fingerprinting tests that can be successfully used is limited, usually to the opened service(s) on the Internet connected system (i.e. TCP port 80). We have already noted that an active operating system fingerprinting tool usually must use several different operating system fingerprinting tests in order to provide with accurate results. With only a limited success of its operating system fingerprinting tests, the quality of the results produced by an active operating system fingerprinting tool will be significantly degraded.

A way to compensate for operating system fingerprinting tests which we would not be able to use in situations that will prove them useless must be found.

Another issue, which needs to be resolved, is the inability to differentiate between different operating systems of the same manufacture.

A part of the remedy is to use application layer based fingerprinting tests tailored towards the services found opened on the targeted system(s) and/or a service commonly found with the operating system family in question. The criteria for adopting one such test, is that it should be hard or impossible to trick the test, and that it will produce accurate results when executed against well fortified Internet connected systems.

Traditional TCP/IP stack based operating system fingerprinting tests must be tailored for maximum impact on the overall fingerprinting results when used in situation in which the targeted system has a limited exposure. The TCP/IP stack based operating system fingerprinting tests to be used must not be easily defeated by commonly found defense systems (i.e. a SYN | FIN scan will be useless when a site is being defended by a stateful inspection-based firewall).

The other part of the remedy is much harder to be implemented.

An active operating system fingerprinting tool should act according to the results its different modules produce. The tool must understand the terrain it is being executed against. If a certain module fails, the fingerprinting tool should conclude what caused the module to fail, and should the tool maintain its current active operating system fingerprinting tactic or switch tactics. If the scanning condition is to be declared hostile, i.e. the target is protected by a filtering device or other means, some fingerprinting modules must not be used, since their effect on the overall test results will result with inaccuracy. Other fingerprinting modules should receive bigger or smaller impact over the overall fingerprinting results according to a certain scheme (i.e. the fingerprinting module is reliable in this particular situation). Certain dependencies must be identified between the different modules of the tool, so modules which depends on the successful execution of another module, will not be executed if the execution of that module had failed.

When possible matches are calculated, the active operating system fingerprinting tool should have intelligence built into the tool allowing it to decide if the received results yields a clear match for an operating system, or that a niche targeted operating system fingerprinting tests tailored towards a common object found with all possible matches (i.e. service, ability, etc.) should be initiated. The niche targeted operating system fingerprinting tests would help to eliminate several possible matches from the list of possible matches found. Hopefully allowing to declare a clear and a decisive match.

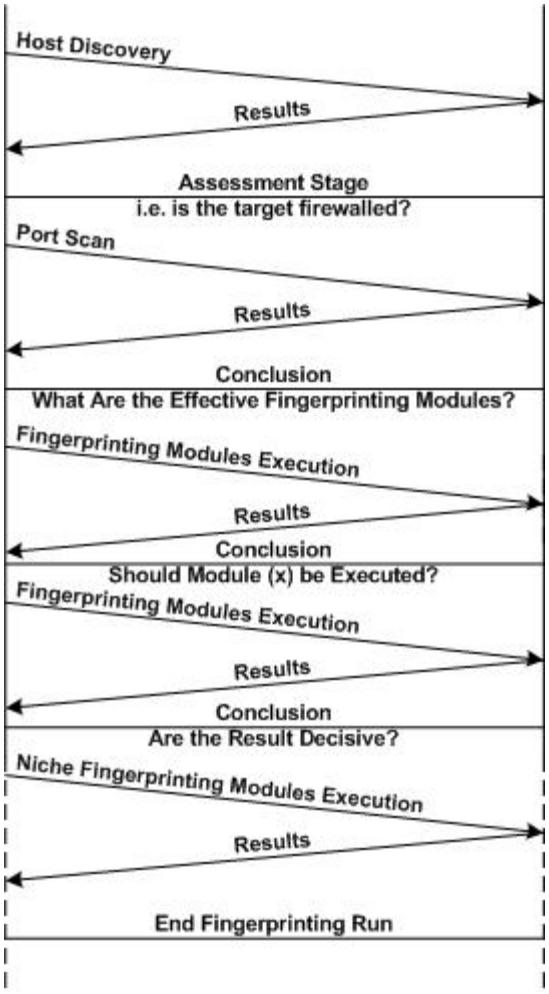

Figure 10: Xprobe2 future run

5.0 Conclusion

Although some advancement was made in the field of operating system fingerprinting in the recent years, still, there are many issues to resolve.

We hope that by enhancing the current and future versions of Xprobe2 we will take active operating system fingerprinting to the next level in its evolution.

6.0 Related Work and Reference

Arkin Ofir, "ICMP Usage in Scanning" research project
http://www.sys-security.com

Arkin Ofir, "ICMP Usage in Scanning" version 3.0, June 2001
http://www.sys-security.com/html/projects/icmp.html

Arkin Ofir & Fyodor Yarochkin, "X – Remote ICMP based OS fingerprinting Techniques", August 2001
http://www.sys-security.com/archive/papers/X_v1.0.pdf

Arkin Ofir & Fyodor Yarochkin, "ICMP based remote OS TCP/IP stack fingerprinting techniques", Phrack Magazine, Volume 11, Issue 57, File 7 of 12, August 11, 2001.
http://www.sys-security.com/archive/phrack/p57-0x07

Arkin Ofir & Fyodor Yarochkin, "Xprobe2 - A 'Fuzzy' Approach to Remote Active Operating System Fingerprinting", http://www.sys-security.com/archive/papers/Xprobe2.pdf, August 2002.

Fyodor, "Remote OS detection via TCP/IP Stack Finger Printing", October 1998
http://www.phrack.com/show.php?p=54&a=9

Franck Veysset, Olivier Courtay, Olivier Heen, "New Tool and Technique for Remote Operating System Fingerprinting" (Response timing calculation based fingerprinting), April 2002
http://www.intranode.com/site/techno/techno_articles.htm