Remote Network Analysis

Torsten Hoefler
htor@cs.tu-chemnitz.de

28th November 2004

Abstract

It is often necessary to collect information about remote systems without having direct shell access to them. Additional knowledge about the system's (operating system and user space) software and the network structure is very helpful for identifying possible attack scenarios which could finally lead to a compromise of the remote system or network. Most systems are accessible from the internet through the IP protocol suite but often protected by a more or less sophisticated firewall/packetfilter. This paper presents a collection of techniques which can be used to map hosts and networks without leaving any traces to Intrusion Detection Systems (IDS).

1 Introduction

Many attack scenarios start with information gaining. There are several techniques to perform this first step. The common goal of these methods is to collect as much information as possible and to do this task as "quite" as possible. Today's most used communication protocol is the Internet Protocol (IP), thus many systems and networks use this protocol for internal and external communication. Many networks are deployed in the same manner, an internal network with full internal connectivity, connected and separated by a firewall from the internet. The firewall permits only outgoing accesses from the internal network. This very easy scenario is shown in Figure 1. If some of the hosts have to serve clients from the internet, another approach has to be taken. This more complicated model is shown in Figure 2. All external servers are deployed in a so called Demilitarized Zone (DMZ), mostly between two packet filters and application gateways. The different possibilities to connect DMZ-hosts are also shown in Figure 2 (dashed lines).

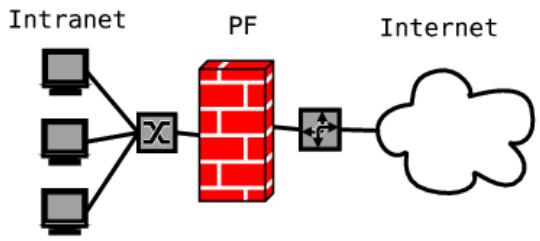

Figure 1: Easy Firewall Model

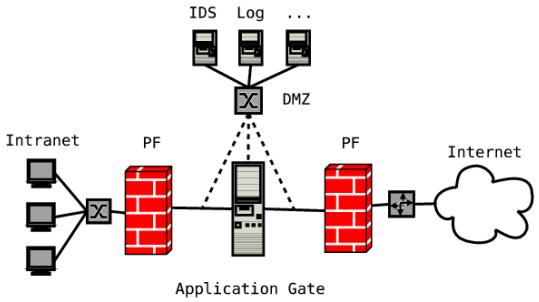

Figure 2: More complex Firewall Model

Generally can be said that internal systems are typically vulnerable to more attacks than external (DMZ) hosts. Network and host mapping can be used to discover ways to reach the internal systems and/or gather additional information about their structure. It is also very useful to discover firewall rules or IP-based access rules for different internal and external hosts. Furtheron, detailed knowledge about the used operating systems or applications can be helpful to find vulnerabilities - the used methodology is called fingerprinting, and should be performed as hidden as possible. One, and maybe the best way to hide fingerprinting attempts is to do nothing but collecting data. If no packet is sent, no packet can trigger an alert. Another idea is to deeply evaluate responses to usual requests (e.g. a HTTP request to a webserver). The packets sent to generate the respose can not be differentiated from normal packets and so never classified as evil. Both ideas are discussed in section 3. But there are several problems with this methodology, the first is that sniffing without doing anything is very hard (nearly impossible) in many scenarios. The second one is the imprecision of the passive approach. The active analysing, described in section 4 addresses this problems and increases the accurancy. The price to pay for this improvement is mainly the loss of "silence" - most active scans are easy to detect and prevent. The legacy active methods are well known and so not discussed in detail. Additionally, there are some new and more sophisticated techniques to fingerprint remote systems. Some of them are discussed in section 5.

2 Related Work

There are many approaches for fingerprinting systems, this paper gives an rough overview about the techniques and provides links and (limited) examples

for tools to apply this knowledge automatically. Main sources and additional information can be found in the References section at the end. The tool synscan [2] tries to combine all of the described methods automatically.

3 Passive Analysis

Passive Analysis is mainly done without sending any packets. Another hidden technique sends normal request packages which can not be differentiated from usual traffic and are so hidden from Intrusion Detection Systems. Thus this approach can also be seen as ”passive”. Analysis can be performed on different layers of the OSI reference model [1]. This paper only deals with layer 2, 3 and 4.

3.1 Layer 2/3

Layer 2 and 3 are implemented in the IP and the TCP[1] for internet communications. These layers are used to administrate virtual connections between arbitrary hosts in a network. The connection management is fully provided by the operating systems TCP/IP stack (commonly used with the sockets api). Thus using fingerprinting techniques on layer 2 or 3 can only provide information about the operating system and the TCP/IP stack.

A commonly used technique for passive fingerprinting of the operating system, called header analysis, is to examine the headers of TCP/IP packets. There are several header values in this packets which are not defined clearly in the RFCs or not implemented correctly by the operating system vendors. Others reveal some information by their definition. Examples for both types are:

- TTL[2] - starts typically with ”defined”, operating system dependent values (e.g. 255, 128, 64) and is decremented by each router on the fly
- Ports - gives rough information about used services and protocols
- DF-Flag - don’t Fragment flag, mostly set by newer operating systems
- Window-Size - amount of data that can be buffered (on the fly at any time)
- TCP-Options - optional tcp-fields, e.g. SACK, Timestamp ...

[1] Transmission Control Protocol
[2] Time To Live

OS	TOS	DF	TTL	Window	Options
Win2000	0	1	128	65535	tsval=0, SACK
Win98	0	1	128	8760	SACK
Linux 2.2	0	1	64	32210	tsval>0, SACK
Linux 2.4	0	1	64	5792	tsval>0, SACK
Linux 2.6	0	1	64	5792	tsval>0, SACK
FreeBSD 4.6	0	1	64	57344	tsval>0
FreeBSD 5.0	0	1	64	65535	tsval>0
OpenBSD 2.x	16	0	64	17520	tsval=0, SACK
...	...	...	...	...	...

Table 1: Header Characteristics for different operating systems

A complete overview about all fields and their average information content regarding to fingerprinting can be seen in figure 3.

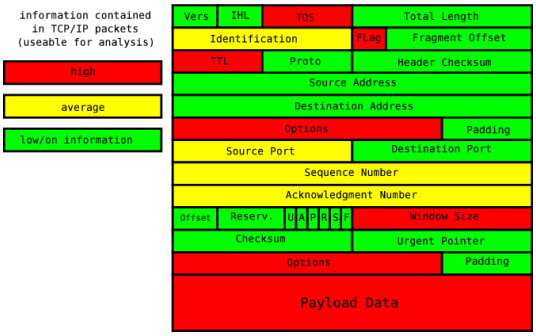

Figure 3: TCP-IP Header Fields and their information content

3.1.1 Example

Table 1 shows different characteristics collected for random operating systems for SYN/ACK packets. Figure 4 shows a SYN/ACK header from the server www.ccc.de collected with a simple *tcpdump* sniffing a normal *wget http://www.ccc.de* . The following header fields are relevant for a classification:

- TOS: 0
- DF: 1
- start TTL: 64 (55)
- Window Size: 65535
- Options: tsval is set, no SACK

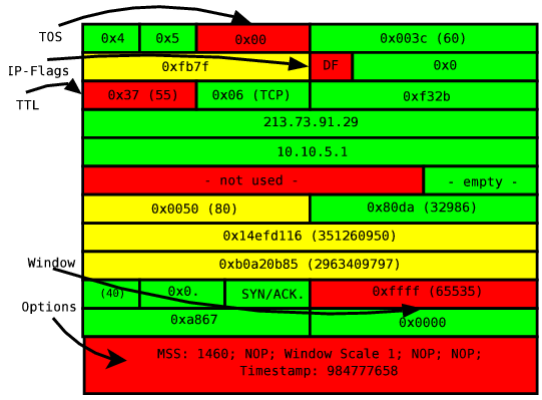

Figure 4: SYN/ACK Header from www.ccc.de

After comparing the collected values to the table, it can be assumed that www.ccc.de runs on FreeBSD 5.0 (is this true?). More examples can be found in the wild. But it is obvious that this methodology is a hard task and can be easily automated. There are several tools like ettercap [3], siphon [4] or p0f [5] which can be used to do the lookup automatically on huge databases. Some of the tools also provide fuzzy matching against the signature database. It is also imaginable to train an artificial intelligence for this task.

But the passive approach is quite inaccurate and very slow. The parameters are often changed by system administrators[3] to improve performance or to prevent fingerprinting. There is also a clear trend noticeable that most modern operating systems behave more and more similar, that a clear distinction between them is hardly possible (e.g. Linux 2.4 vs. Linux 2.6).

3.1.2 Summary

Layer 2/3 fingerprinting utilizes the imprecise standard definition or incorrect TCP/IP stack implementations to differ between operating systems. The currently used signature-database approach has several disadvantages and can be improved by using fuzzy matching or artificial intelligence. The precision of remote OS fingerprinting can be enhanced by using active fingerprinting methods.

3.2 Layer 4

Passive methods for Layer 4 application analysis can use the same approach for TCP/IP payload, as shown in chapter 3.1 for header data. Due to the fact that there is no standardized "payload-protocol", this task has to be done for every layer 4 protocol separately.

[3] either at the host system or on intermediate routers/gateways

There is currently no automated (passive) tool for analyzing the data-streams to the autors knowledge.

4 Active Analysis

Active analysing is mainly done by sending specially crafted requests and evaluating the responses. This offers much more flexibility, the packets can be sent in any way, provoking failures and recording repsonses. Even non-protocol confirm messages can be sent to the victim. This analysis can also be done on all different layers, layer 2, 3 and 4 are discussed in the following sections.

4.1 Layer 2/3

Layer 2/3 fingerprinting is as described in section 3.1 targeted at the TCP/IP stack and operating system fingerprinting. Well known techniques are used to query the remote TCP/IP stack under different conditions (e.g. NULL packet to a open port, ACK packet to a open port, SYN, ACK, and FIN|PSH|URG to a closed port ...). These special crafted packets are easily to detect in a TCP/IP stream and can so be blocked[4] or modified[5] by firewalls/gateways. The commonly used and most advanced tool for performing active OS detection seems to be nmap[6] by Fyodor [6]. It combines a lot of different techniques (e.g. FIN to open port, ISN Sampling, ICMP Tests, TCP Options, Fragmentation Handling) to achieve best results. But due to its popularity, the packets it sends are recognized by most Intrusion Detection Systems and can be blocked easily. The second drawback is that nmap needs one opened and one closed TCP port and one closed UDP port to perform fingerprinting. This port constellation is mostly not available if firewalls (even personal firewalls) are used to protect the system. Xprobe2 [7], a nmap-like tool to perform fuzzy matching is also available but has similar drawbacks. The methodology is described in more detail by Fyodor in [8].

4.2 Layer 4

There are several techiques available for active fingerprinting level 4 applications. The maybe oldest is the banner grabbing, which can still be quite interesting for application information gathering. An easy example with netcat ([9]) is given in the following:

[4] e.g. implicit with stateful firewalling
[5] e.g. change TTL,TOS or TCP-Options with iptables
[6] command line option: -O

```
htor@archimedes:~$ echo -e "HEAD / HTTP\
/1.0\n\n" | nc www.example.de 80
HTTP/1.1 404 Not Found
Date: Sat, 20 Nov 2004 19:54:52 GMT
Server: Apache/1.3.29 (Unix)
Connection: close
Content-Type: text/html;
```

The maybe easiest way to find the protocol spoken behind a specific port is to take a look in the IANA assigned port number list [10].

Another approach to gain knowledge about the operating system is the binary analysis of FTP servers (if the software version is not available by banner grabbing). An administrative binary can be downloaded and analyzed for its format. An example follows:

```
htor@archimedes:~$ wget ftp://ftp.ub.\
uni-bielefeld.de/bin/ls --passive \
-o /dev/null
htor@archimedes:~$ file ls
ls: ELF 32-bit MSB executable, SPARC,
version 1 (SYSV), dynamically linked
(uses shared libs), stripped
```

If these techniques are not applicable, there is another more sophisticated approach by sending different protocol "hello"'s and evaluating the answer for guessing the protocol of the application behind a specific port. After this step, one could send several special crafted protocol requests and generate error responses to conclude the application type or version number. This task is very hard to perform manually, therefore are tools available. Nmap [6] supports application fingerprinting with its "-sV" switch and the tool amap [11] is also able to perform this kind of scanning.

5 Advanced Methods

This section discusses several techniques which can be used to perform advanced hidden scanning or fingerprinting. Most of them are hardly detectable and/or not backtraceable.

5.1 Old Techniques

The old techniques include the inverse mapping, which sends normal inconspicious packets to hosts behind firewalls. This method is only applicable for stateless firewalls and IDS. The last router will respond a ICMP host unreachable packet for every non-existing host[7], enabling the attacker to generate a detailed network map. Another idea is a so called slow scan: the packets are sent with a very low frequency (e.g. < 1 packet / hour) that an IDS does not observe an attack.

[7] most routers determine the existence of a host by the ARP protocol

5.2 Remote Identification Next Generation

The RING algorithm, proposed in [12], measures the TCP retransmission-times and -count and utilizes the deviations to distinct between different TCP/IP stacks. A single SYN packet is sent to the victim and nothing else (no response to SYN/ACK packets). The count of and the times between the arriving SYN/ACK packets is measured and compared to a database. The scheme is shown in Figure 5. Automated tools are available with snacktime [13] and Cron-OS [14].

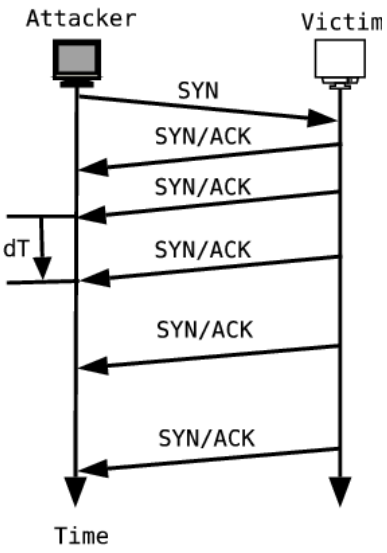

Figure 5: RING working principle

5.3 Idle Scan

Idle Scanning uses a "zombie"-host for scanning. All packets are sent with the "zombie"-host as source and a potential weakness in the zombies IP-Stack implementation (predictalbe IP-IDs) is used to determine the result. So the IDS and firewalls at the target network see the "zombie" as requester (attacker). Thus this technique is also useable to determine IP-based filter rules (if the IP to test has predictable IP-IDs). Four steps have to be performed:

1. find a zombie (predictable IP-ID and low current traffic)

2. determine the IP-ID of the zombie
3. send spoofed packet to target
4. query zombie for IP-ID

The whole process is shown in Figure 6. Automation is provided by nmap [6] with the "-D" command-line switch.

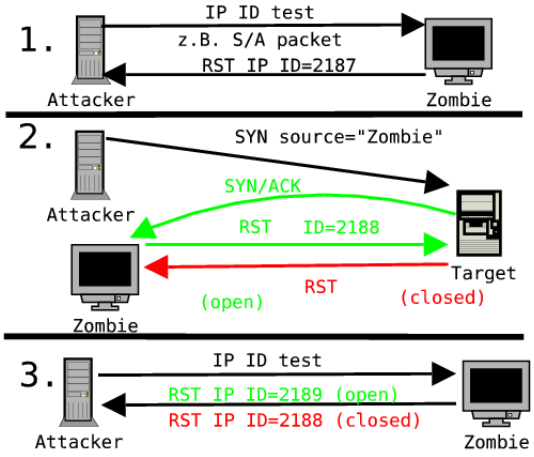

Figure 6: Idle Scan working principle

5.4 Firewalking

Firewalking can be seen as a technique to traverse firewalls. This task is quite easy for stateless firewalls - just send legal packets which are allowed by the filter rules (mainly ACK, SYN/ACK, FIN ...) and evaluate the response. There is no commmon scheme to analyse the response packets, but they often contain a lot of information about the network (e.g. ICMP Host Unreachable, ICMP Network Unreachable ...).

Another definition for the term firewalking is given by the Cambridge Technology Partners as "A Traceroute-Like Analysis of IP Packet Responses to determine Gateway Access Control Lists". This new technique uses firewall traversing to determine a ruleset without sending any packet to the target system. It is mainly done in two steps:

1. determine hop-count to firewall (= HC(FW))
2. send packets with TTL=HC(FW)+1 with SYN flag set

⇒ if the target host is more than 1 hop behind the gateway, the packet will never reach it (the next router will drop it and respond with ICMP Time exceed). This only happens if the firewall has permitted the connection. If there is a prohibiting firewall rule, there will be no or another response. This technique is not very relieable because the prevention is straightforward: drop outgoing ICMP time exceed packets.

6 Overview Fingerprinting

Figure 7 gives a slight overview about the different fingerprinting techniques and some examples for tools which can be used for automatic analysis.

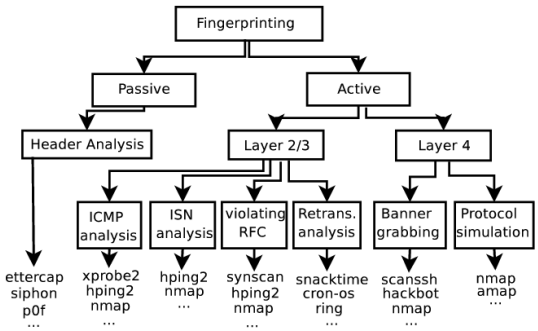

Figure 7: Overview about fingerprinting

References

[1] OSI GROUP: *The OSI Reference Model (see: http://en.wikipedia.org/wiki/OSI_model)*

[2] SYNSCAN *http://synscan.sourceforge.net/*

[3] ETTERCAP *http://ettercap.sourceforge.net/*

[4] SIPHON *http://siphon.datanerds.net/*

[5] P0F *http://lcamtuf.coredump.cx/p0f.shtml*

[6] NMAP *http://www.insecure.org/nmap/*

[7] XPROBE2 *http://sourceforge.net/projects/xprobe/*

[8] FYODOR *Remote OS detection via TCP/IP Stack FingerPrinting*

[9] NETCAT *http://netcat.sourceforge.net/*

[10] IANA ASSIGNED PORT NUMBERS *http://www.iana.org/assignments/port-numbers*

[11] AMAP *http://www.thc.org/releases.php*

[12] RING *http://www.intranode.com/pdf/techno/ring-full-paper.pdf*

[13] SNACKTIME *http://www.planb-security.net/wp/snacktime.html*

[14] CRON-OS *http://home.gna.org/cronos/*