

# Nmap Service & Application Version Detection

By Fyodor

## Introduction

While Nmap does many things, its most fundamental feature is port scanning. Point Nmap at a remote machine, and it might tell you that ports 25/tcp, 80/tcp, and 53/udp are open. Using its nmap-services database of more than 2,200 well-known services, Nmap would report that those ports probably correspond to a mail server (SMTP), web server (HTTP), and name server (DNS) respectively. This lookup is usually accurate -- the vast majority of daemons listening on TCP port 25 are, in fact, mail servers. However, you should not bet your security on this! People can and do run services on strange ports. Perhaps their main web server was already on port 80, so they picked a different port for a staging or test server. Maybe they think hiding a vulnerable service on some obscure port prevents "evil hackers" from finding it. Even more common lately is that people choose ports based not on the service they want to run, but on what gets through the firewall. When ISPs blocked port 80 after major Microsoft IIS worms CodeRed and Nimda, hordes of users responded by moving their personal web servers to another port. When companies block telnet access due to its horrific security risks, I have seen users simply run telnetd on the Secure Shell (SSH) port instead.

Even if Nmap is right, and the hypothetical server above is running SMTP, HTTP, and DNS servers, that is not a lot of information. When doing vulnerability assessments (or even simple network inventories) of your companies or clients, you really want to know which mail and DNS servers and versions are running. Having an accurate version number helps dramatically in determining which exploits a server is vulnerable to. Do keep in mind that security fixes are often backported to earlier versions of software, so you cannot rely solely on the version number to prove a service is vulnerable.

Another good reason for determining the service types and version numbers is that many services share the same port number. For example, port 258/tcp is used by both the Checkpoint Firewall-1 GUI management interface and the yak Windows chat client. This makes a guess based on the nmap-services table even less accurate. Anyone who has done much scanning knows that you also often find services listening on unregistered ports - these are a complete mystery without version detection. A final problem is that filtered UDP ports often look the same to a simple port scanner as open ports (see Chapter 3 - Mainstream Port Scanning Techniques). But if they respond to the service-specific probes sent by Nmap version detection, you know for sure that they are open (and often exactly what is running).

The Nmap version scanning subsystem (introduced in version 3.45) tries to answer all these questions by connecting to open ports and interrogating them for further information using probes that the specific services understand. This allows Nmap to give a detailed assessment of what is really running, rather than just what port numbers are open. Example 1, "Simple usage of version detection" shows the actual output.

### Example 1. Simple usage of version detection

```
# nmap -A -T4 -F www.insecure.org

Starting nmap 3.40PVT16 ( http://www.insecure.org/nmap/ )
Interesting ports on www.insecure.org (205.217.153.53):
(The 1206 ports scanned but not shown below are in state: filtered)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 3.1p1 (protocol 1.99)
25/tcp    open  smtp     Qmail smtpd
53/tcp    open  domain   ISC Bind 9.2.1
80/tcp    open  http     Apache httpd 2.0.39 ((Unix) mod_perl/1.99_07-dev Perl/v5.6.1)
113/tcp   closed auth
```

# Nmap Service & Application Version Detection

By Fyodor

```
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux Kernel 2.4.0 - 2.5.20
Uptime 108.307 days (since Wed May 21 12:27:44 2003)
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 34.962 seconds
```

Nmap version detection offers the following advanced features (fully described later):

- High speed, parallel operation via non-blocking sockets and a probe/match definition grammar designed for efficient yet powerful implementation.
- Determines the application name and version number where available -- not just the service protocol.
- Supports both the TCP and UDP protocols, as well as both textual ASCII and packed binary services.
- Multi-platform support, including Linux, Windows, Mac OS X, FreeBSD/NetBSD/OpenBSD, Solaris, and all the other platforms on which Nmap is known to work.
- If SSL is detected, Nmap connects using OpenSSL (if available) and tries to determine what service is listening behind that encryption layer. This allows it to discover services like https, pop3s, imaps, etc. as well as providing version details.
- If a SunRPC service is discovered, Nmap launches its brute-force RPC grinder to find the program number, name, and version number.
- IPv6 is supported, including TCP, UDP, and SSL over TCP.
- Community contributions - If Nmap gets data back from a service that it does not recognize, a "service fingerprint" is printed along with a submission URL. This system is patterned after the extremely successful Nmap OS Detection (Chapter 7) fingerprint submission process. New probes and corrections can also be submitted.
- Comprehensive database - Nmap recognizes more than one thousand service signatures, covering more than 180 unique service protocols from acap, afp, and aim to xml-rpc, zebedee, and zebra.

## Usage/Examples

Before delving into the technical details of how version detection is implemented, here are some examples demonstrating its usage and capabilities. To enable version detection, just add `-sV` to whatever Nmap flags you normally use. Or use the `-A` option, which also turns on OS detection (`-O`, Chapter 7) and may enable other advanced and aggressive features later. It is really that simple, as shown in Example 2, "Version detection against WWW.Microsoft.Com".

### Example 2. Version detection against WWW.Microsoft.Com

```
# nmap -A -T4 -F www.microsoft.com

Starting nmap 3.40PVT16 ( http://www.insecure.org/nmap/ )
Interesting ports on 80.67.68.30:
(The 1208 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE VERSION
22/tcp open  ssh      Akamai-I SSH (protocol 1.5)
80/tcp open  http     AkamaiGHost (Akamai's HTTP Acceleration/Mirror service)
443/tcp open  ssl/http AkamaiGHost (Akamai's HTTP Acceleration/Mirror service)
Device type: general purpose
Running: Linux 2.1.X|2.2.X
```

# Nmap Service & Application Version Detection

By Fyodor

```
OS details: Linux 2.1.19 - 2.2.25
Uptime 22.924 days (since Fri Aug 15 03:34:27 2003)
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 19.223 seconds
```

This preceding scan demonstrates a couple things. First of all, it is gratifying to see WWW.Microsoft.Com served off one of Akamai's Linux boxes. More relevant to this chapter is that the "service" for port 443 is "ssl/http". That means that service detection first discovered that the port was SSL, then it loaded up OpenSSL and performed service detection again through SSL connections to discover a web server running AkamiGHost behind the encryption. Recall that -T4 causes Nmap to go faster (more aggressive timing) and -F tells Nmap to scan only ports registered in nmap-services.

Example 3, "Complex version detection" is a longer and more diverse example.

## Example 3. Complex version detection

```
./nmap -A -T4 localhost

Starting nmap 3.40PVT16 ( http://www.insecure.org/nmap/ )
Interesting ports on felix (127.0.0.1):
(The 1640 ports scanned but not shown below are in state: closed)
PORT      STATE SERVICE  VERSION
21/tcp    open  ftp      WU-FTPD wu-2.6.1-20
22/tcp    open  ssh      OpenSSH 3.1p1 (protocol 1.99)
53/tcp    open  domain   ISC Bind 9.2.1
79/tcp    open  finger   Linux fingerd
111/tcp   open  rpcbind  2 (rpc #100000)
443/tcp   open  ssl/http Apache httpd 2.0.39 ((Unix) mod_perl/1.99_04-dev [cut])
515/tcp   open  printer
631/tcp   open  ipp      CUPS 1.1
953/tcp   open  rndc?
5000/tcp  open  ssl/ftp  WU-FTPD wu-2.6.1-20
5001/tcp  open  ssl/ssh  OpenSSH 3.1p1 (protocol 1.99)
5002/tcp  open  ssl/domain ISC Bind 9.2.1
5003/tcp  open  ssl/finger Linux fingerd
6000/tcp  open  X11      (access denied)
8000/tcp  open  http-proxy Junkbuster webproxy
8080/tcp  open  http     Apache httpd 2.0.39 ((Unix) mod_perl/1.99_04-dev [cut])
8081/tcp  open  http     Apache httpd 2.0.39 ((Unix) mod_perl/1.99_04-dev [cut])
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux Kernel 2.4.0 - 2.5.20
Uptime 8.653 days (since Fri Aug 29 11:16:40 2003)

Nmap run completed -- 1 IP address (1 host up) scanned in 42.494 seconds
```

You can see here the way RPC services are treated, with the brute force RPC scanner being used to determine that port 111 is rpcbind version 2. You may also notice that port 515 gives the service as "printer", but that version column is empty. This means that Nmap did determine the service name via its probing, but was not able to determine anything else. On the other hand, port 953 gives the service as "rndc?". The question mark tells us that Nmap was not even able to determine the service name through probing. As a fallback, rndc is mentioned because that has port 953 registered in nmap-

# Nmap Service & Application Version Detection

By Fyodor

services. Unfortunately, none of Nmap's probes elicited any sort of response from rndc. If they had, Nmap would have printed a service fingerprint and a submission URL so that it could be recognized in the next version. As it is, Nmap requires a special probe. One might even be available by the time you read this. The upcoming "community contributions" section provides details on writing your own probes.

It is also worth noting that some services provide much more information than just the version number. Examples above include whether X11 permits connections, the SSH protocol number, and the Apache module versions list. Some of the Apache modules even had to be cut from the output to fit on this page.

A few early reviewers questioned the sanity of running services such as SSH and finger over SSL. This was actually just fun with [stunnel](#), in part to ensure that parallel SSL scans actually work.

## Technique Described

Nmap version scanning is actually rather straightforward. It was designed to be as simple as possible while still being scalable, fast, and accurate. The truly nitty-gritty details are best discovered by downloading and reviewing the source code, but a synopsis of the techniques used follows.

Nmap first does a port scan as per your instructions, and then passes all the open TCP and/or UDP ports to the service scanning module. Those ports are then interrogated in parallel, although a single port is described here for simplicity.

1. Nmap checks to see if the port is one of the ports to be excluded, as specified by the Exclude directive in nmap-service-probes. If it is, Nmap will not scan this port for reasons mentioned in the section called "[nmap-service-probes](#) File Format".
2. If the port is TCP, Nmap starts by connecting to it.
3. Once the TCP connection is made, Nmap listens for roughly 5 seconds. Many common services, including most ftp, ssh, smtp, telnet, pop3, and imap servers, identify themselves in an initial welcome banner. Nmap refers to this as the "NULL probe", because Nmap just listens for responses without sending any probe data. If any data is received, Nmap compares it to hundreds of signature regular expressions in its nmap-service-probes file (described in the section called "[nmap-service-probes](#) File Format"). If the service is fully identified, we are done with that port! The regular expression includes substrings that can be used to pick version numbers out of the response. In some cases, Nmap gets a "soft match" on the service type, but no version info. In that case, Nmap continues but only sends probes that are known to recognize the soft-matched service type.
4. At this point, Nmap UDP probes start, and TCP connections end up here if the NULL probe above fails or soft-matches. Since the reality is that most ports are used by the service they are registered to in nmap-services, every probe has a list of port numbers that are considered "good bets". For example, the probe called GetRequest that recognizes web servers (among other services) lists 80-85, 8000-8010, and 8080-8085 as probable ports. Nmap sequentially executes the probe(s) that match the port number being scanned. Each probe includes a probe string (which can be arbitrary ASCII text or \xHH escaped binary), which is sent to the port. Responses that come back are compared to a list of regular expressions of the same type as discussed in the NULL probe description above. As with the NULL probe, these tests can either result in a full match (ends processing for the remote service), a soft match (limits future probes to those which match a certain service), or no match at all. The exact list of regular expressions that Nmap uses to test for a match depends on the probe fallback configuration. For instance, the data returned from the X11Probe is very unlikely to match any regular expressions crafted for the GetRequest probe. On the other hand, it is likely that results returned from a Probe such as RTSPRequest

# Nmap Service & Application Version Detection

By **Fyodor**

might match a regular expression crafted for GetRequest since the 2 protocols being tested for are closely related. So the RTSPRequest probe has a fallback to GetRequest matches. For a more comprehensive explanation, see the section called "Cheats and Fallbacks".

5. In most cases, the "NULL probe" or the probable port probe(s) (there is usually only one) described above matches the service. Since the NULL "probe" shares its connection with the probable port probe, this allows service detection to be done with only one brief connection in most cases. With UDP only one packet is usually required. But should the NULL probe and probable port probe(s) fail, Nmap goes through all of the existing probes sequentially. In the case of TCP, Nmap must make a new connection for each probe to avoid having previous probes corrupt the results. This worst-case scenario can take a bit of time, especially since Nmap must wait about five seconds for the results from each probe because of slow network connections and otherwise slowly responding services. Fortunately, Nmap utilizes several automatic techniques to speed up scans:
  - Nmap makes most probes generic enough to match many services. For example, the GenericLines probe sends two blank lines ("`\r\n\r\n`") to the service. This matches daemons of many diverse service types, including ftp, ident, pop3, uucp, postgres, and whois. The GetRequest probe matches even more service types. Other examples include "`help\r\n`" and generic RPC and MS SMB probes.
  - If a service matches a softmatch directive, Nmap only needs to try probes that can potentially match that service.
  - All probes were not created equal! Some match many more services than others. Because of this, Nmap uses the rarity metric to avoid trying probes that are extremely unlikely to match. Experienced Nmap users can force all probes to be tried regardless or limit probe attempts even further than the default by using the `--version_intensity`, `--version_all`, and `--version_light` options discussed in the section called "Probe Selection and Rarity".
6. One of the probes tests whether the target port is running SSL. If so (and if OpenSSL is available), Nmap connects back via SSL and restart the service scan to determine what is listening behind the encryption. A special directive allows different probable ports for normal and SSL tunneled connections. For example, Nmap should start against port 443 (https) with an SSL probe. But after SSL is detected and enabled, Nmap should try the GetRequest probe against port 443 because that port usually has a web server listening behind SSL encryption.
7. Another generic probe identifies RPC-based services. When these are found, the Nmap RPC Grinder (discussed later) is initiated to brute force the RPC program number/name and supported version numbers. Similarly, an SMB postprocessor for fingerprinting Windows services may be added eventually.
8. If at least one of the probes elicits some sort of response, yet Nmap is unable to recognize the service, the response content is printed to the user in the form of a "fingerprint" that can be submitted at a provided URL for the next version of Nmap.

## Cheats and Fallbacks

Even though Nmap waits a generous amount of time for services to reply, sometimes an application is slow to respond to the NULL probe. This can occur for a number of reasons, including slow reverse DNS lookups performed by some services. Because of this, Nmap can sometimes match the results from a subsequent probe to a match line designed for the NULL probe.

For example, suppose we scan port 25 (smtp) on a server to determine what is listening. As soon as we connect, that service may conduct a bunch of DNS blacklist lookups to determine whether we should be treated as spammers and denied service. Before it finishes that, Nmap gives up waiting for a NULL probe response and sends the next probe with port 25 registered, which is "`HELP\r\n`". When the service finally completes its anti-spam checks, it prints a greeting banner, reads the Help probe, and responds as shown in Example 4, "NULL Probe Cheat Example Output".

# Nmap Service & Application Version Detection

By Fyodor

## Example 4. NULL Probe Cheat Example Output

```
220 hcs.w.org ESMTP Sendmail 8.12.3/8.12.3/Debian-7.1; Tue, [cut]
214-2.0.0 This is sendmail version 8.12.3
214-2.0.0 Topics:
214-2.0.0 HELO EHLO MAIL RCPT DATA
214-2.0.0 RSET NOOP QUIT HELP VRFY
214-2.0.0 EXPN VERB ETRN DSN AUTH
214-2.0.0 STARTTLS
214-2.0.0 For more info use "HELP <topic>".
214-2.0.0 To report bugs in the implementation send email to
214-2.0.0 sendmail-bugs@sendmail.org.
214-2.0.0 For local information send email to Postmaster at your site.
214 2.0.0 End of HELP info
```

Nmap reads this data from the socket and finds that no regular expressions from the Help probe match the data returned. This is because Nmap normally expects to receive the ESMTP banner during the NULL probe and match it there.

Because this is a relatively common scenario, Nmap “cheats” by trying to match responses to any of the NULL Probe match lines if none of the probe-specific lines match. In this case, a NULL match line exists which reports that the program is Sendmail, the version is 8.12.3/8.12.3/Debian-7.1, and the hostname is hcs.w.org.

The NULL probe cheat is actually just a specific example of a more general Nmap feature: fallbacks. The fallback directive is described in detail in the section called “[nmap-service-probes](#) File Format”. Essentially, any probe that is likely to encounter results that can be matched by regular expressions in other probes has a fallback directive that specifies these other probes.

For example, in some configurations of the popular Apache web server, Apache won't respond to the GetRequest (“GET / HTTP/1.0\r\n\r\n”) probe because no “virtual host” has been specified. Nmap is still able to correctly identify these servers because those servers usually respond to the HTTPOptions probe. That probe has a fallback to the GetRequest regular expressions, which are sufficiently general to recognize Apache's responses to the HTTPOptions probes.

## Probe Selection and Rarity

In determining what probes to use, Nmap considers their “rarity”. This is an indication of how likely the probe is to return useful data. If a probe has a high rarity, it is considered less common and is less likely to be tried. Nmap users can specify which probes are tried by changing the “intensity” level of the version scan, as described below. The precise algorithm Nmap uses when determining which probes to use follows:

1. For TCP, the NULL probe is always tried first.
2. All probes that have the port being scanned listed as a probable port (see the section called “[nmap-service-probes](#) File Format”) are tried in the order they appear in nmap-service-probes.
3. All other probes that have a rarity value less than or equal to the current intensity value of the scan are tried, also in the order they appear in nmap-service-probes.

Once a probe is found to match, the algorithm terminates and results are reported.

# Nmap Service & Application Version Detection

By Fyodor

Because all of Nmap's probes have a rarity value associated with them, it is relatively easy to control how many of them are tried when performing a version scan. Simply choose an intensity level appropriate for a scan. The higher an intensity level, the more probes will be tried. So if a very comprehensive scan is desired, a high intensity level is appropriate - even though it may take longer than a scan conducted at a lower intensity level. Nmap's default intensity level is 7 but Nmap provides the following switches for different scanning needs:

## **--version\_intensity**

Syntax: `--version_intensity <intensity level between 0 and 9>`

Example:

```
nmap -sV --version_intensity 3 scanme.nmap.org
```

Sets the intensity level of a version scan to the specified value. If 0 is specified, only the NULL probe (for TCP) and probes that list the port as a probable port are tried.

## **--version\_light**

Syntax: `--version_light`

Example:

```
nmap -sV --version_light host.com
```

Sets the intensity level to 2.

## **--version\_all**

Syntax: `--version_all`

Example:

```
nmap -sV --version_all host.com
```

Sets the intensity level to 9. Since all probes have a rarity level between 1 and 9, this tries all of the probes.

## **Technique Demonstrated**

If the English description above is not clear enough, you can see for yourself how it works by adding the `--version_trace` (and usually `-d` (debugging)) options to your Nmap command line. This shows all the connection and data read/write activity of the service scan. Example 5, "Detailed trace of version detection" is a real annotated example (output slightly modified for readability).

# Nmap Service & Application Version Detection

By Fyodor

## Example 5. Detailed trace of version detection

```
# nmap -sSV -T4 -F -d --version_trace www.insecure.org
```

```
Starting nmap 3.48 ( http://www.insecure.org/nmap/ )
Host www.insecure.org (205.217.153.53) appears to be up ... good.
Initiating SYN Stealth Scan against www.insecure.org (205.217.153.53) at 19:53
Initiating service scan against 4 services on 1 host at 19:53
```

*The SYN scan has found 4 open ports - now we are beginning a service scan against each of them in parallel. We start with a TCP connection for the NULL probe:*

```
Starting probes against new service: 205.217.153.53:22 (tcp)
NSOCK (2.0750s) TCP connection requested to 205.217.153.53:22 (IOD #1) EID 8
Starting probes against new service: 205.217.153.53:25 (tcp)
NSOCK (2.0770s) TCP connection requested to 205.217.153.53:25 (IOD #2) EID 16
Starting probes against new service: 205.217.153.53:53 (tcp)
NSOCK (2.0830s) TCP connection requested to 205.217.153.53:53 (IOD #3) EID 24
Starting probes against new service: 205.217.153.53:80 (tcp)
NSOCK (2.0860s) TCP connection requested to 205.217.153.53:80 (IOD #4) EID 32
NSOCK (2.0870s) Callback: CONNECT SUCCESS for EID 32 [205.217.153.53:80]
NSOCK (2.0870s) Read request from IOD #4 [205.217.153.53:80] (timeout: 5000ms) EID 42
NSOCK (2.0870s) Callback: CONNECT SUCCESS for EID 24 [205.217.153.53:53]
NSOCK (2.0870s) Read request from IOD #3 [205.217.153.53:53] (timeout: 5000ms) EID 50
NSOCK (2.0870s) Callback: CONNECT SUCCESS for EID 16 [205.217.153.53:25]
NSOCK (2.0870s) Read request from IOD #2 [205.217.153.53:25] (timeout: 5000ms) EID 58
NSOCK (2.0870s) Callback: CONNECT SUCCESS for EID 8 [205.217.153.53:22]
NSOCK (2.0870s) Read request from IOD #1 [205.217.153.53:22] (timeout: 5000ms) EID 66
```

*At this point, "Null probe" connections have successfully been made to all four services. It starts at 2 seconds because that is how long the ping and SYN scans took.*

```
NSOCK (2.0880s) Callback: READ SUCCESS for EID 66 [205.217.153.53:22] (23 bytes): SSH-1.99-
OpenSSH_3.1p1.
Service scan match: www.insecure.org (205.217.153.53):22 is ssh. Version:
|OpenSSH|3.1p1|protocol 1.99|
```

*SSH was nice enough to fully identify itself immediately upon connection as OpenSSH 3.1p1. One down, three to go.*

```
NSOCK (2.0880s) Callback: READ SUCCESS for EID 58 [205.217.153.53:25] (27 bytes): 220
core.lnxnet.net ESMTP..
Service scan soft match: www.insecure.org (205.217.153.53):25 is smtp
```

*The mail server on port 25 also gave us a useful banner. We do not know what type of mail server it is, but starting with "220 " and including the word "ESMTP" tells us it is a mail (SMTP) server. So Nmap softmatches smtp, meaning that only probes able to match SMTP servers are tried from now on. Note that non-printable characters are represented by dots -- so the "." after ESMTP is really the "\r\n" line termination sequence.*

# Nmap Service & Application Version Detection

By Fyodor

```
NSOCK (2.0880s) Read request from IOD #2 [205.217.153.53:25] (timeout: 4996ms) EID 74
NSOCK (7.0880s) Callback: READ TIMEOUT for EID 74 [205.217.153.53:25]
NSOCK (7.0880s) Write request for 6 bytes to IOD #2 EID 83 [205.217.153.53:25]: HELP..
NSOCK (7.0880s) Read request from IOD #2 [205.217.153.53:25] (timeout: 5000ms) EID 90
```

*Nmap listens a little longer on the SMTP connection, just in case the server has more to say. The read request times out after 5 seconds. Nmap then finds the next probe which is registered to port 25 and has smtp signatures. That probe simply consists of "HELP\r\n", which Nmap writes into the connection.*

```
NSOCK (7.0880s) Callback: READ TIMEOUT for EID 50 [205.217.153.53:53]
NSOCK (7.0880s) Write request for 32 bytes to IOD #3 EID 99 [205.217.153.53:53]:
.....version.bind.....
NSOCK (7.0880s) Read request from IOD #3 [205.217.153.53:53] (timeout: 5000ms) EID 106
```

*The DNS server on port 53 does not return anything at all. The first probe registered to port 53 in nmap-service-probes is DNSVersionBindReq, which queries a DNS server for its version number. This is sent onto the wire.*

```
NSOCK (7.0880s) Callback: READ TIMEOUT for EID 42 [205.217.153.53:80]
NSOCK (7.0880s) Write request for 18 bytes to IOD #4 EID 115 [205.217.153.53:80]: GET /
HTTP/1.0....
NSOCK (7.0880s) Read request from IOD #4 [205.217.153.53:80] (timeout: 5000ms) EID 122
```

*The port 80 NULL Probe also failed to return any data. An HTTP GET request is sent, since that probe is registered to port 80.*

```
NSOCK (7.0920s) Callback: READ SUCCESS for EID 122 [205.217.153.53:80] [EOF](15858 bytes)
Service scan match: www.insecure.org (205.217.153.53):80 is http. Version: |Apache
httpd|2.0.39|(Unix) mod_perl/1.99_07-dev...
```

*Apache returned a huge (15KB) response, so it is not printed. That response provided detailed configuration information, which Nmap picks out of the response. There are no other probes registered for port 80. So if this had failed, Nmap would have tried the first TCP probe in nmap-service-probes. That probe simply sends blank lines ("r\nr\n"). A new connection would have been made in case the GET probe confused the service.*

```
NSOCK (7.0920s) Callback: READ SUCCESS for EID 106 [205.217.153.53:53] (50 bytes):
.0.....version.bind.....9.2.1
Service scan match: www.insecure.org (205.217.153.53):53 is domain. Version: ||ISC Bind|9.2.1||
```

*Port 53 responded to our DNS version request. Most of the response (as with the probe) is binary, but you can clearly see the version 9.2.1 there. If this probe had failed, the next probe registered to port 53 is a DNS server status request (14 bytes: "\0x0C\0\0x10\0\0\0\0\0\0\0\0"). Having this backup probe helps because many more servers respond to a status request than a*

# Nmap Service & Application Version Detection

By Fyodor

*version number request.*

NSOCK (7.0920s) Callback: READ SUCCESS for EID 90 [205.217.153.53:25] (55 bytes): 214 qmail home page: http...

Service scan match: www.insecure.org (205.217.153.53):25 is smtp. Version: |qmail smtpd|||

*Port 25 gives a very helpful response to the "Help" probe. Other SMTP servers such as Postfix, Courier, and Exim can often be identified by this probe as well. If the response did not match, Nmap would have given up on this service because it had already softmatched smtp and there are no more smtp probes in nmap-service-probes.*

The service scan took 5 seconds to scan 4 services on 1 host.

*This service scan run went pretty well. No service required more than one connection. It took five seconds because Qmail and Apache hit the 5-second NULL probe timeout before Nmap sent the first real probes. Here is the reward for these efforts:*

Interesting ports on www.insecure.org (205.217.153.53):

(The 1212 ports scanned but not shown below are in state: closed)

PORT STATE SERVICE VERSION

22/tcp open ssh OpenSSH 3.1p1 (protocol 1.99)

25/tcp open smtp qmail smtpd

53/tcp open domain ISC Bind 9.2.1

80/tcp open http Apache httpd 2.0.39 ((Unix) mod\_perl/1.99\_07-dev Perl/v5.6.1)

Nmap run completed -- 1 IP address (1 host up) scanned in 7.104 seconds

## Post-processors

Nmap is usually finished working on a port once it has deduced the service and version information as demonstrated above. However, there are certain services for which Nmap performs additional work. These post-processors are presently available for RPC and SSL services, and Windows SMB interrogation is under consideration.

## RPC Grinding

SunRPC (Sun Remote Procedure Call) is a common UNIX protocol used to implement many services including NFS. Nmap ships with an nmap-rpc database of almost 600 RPC programs. Many RPC services use high-numbered ports and/or the UDP transport protocol, making them available through many poorly configured firewalls. RPC programs (and the infrastructure libraries themselves) also have a long history of serious remotely exploitable security holes. So network admins and security auditors often wish to learn more about any RPC programs on their networks.

If the portmapper (rpcbind) service (UDP or TCP port 111) is available, RPC services can be enumerated with the UNIX rpcinfo command. Example 6, "Enumerating RPC services with rpcinfo" demonstrates this against a default Solaris 9 server.

## Example 6. Enumerating RPC services with rpcinfo

# Nmap Service & Application Version Detection

By Fyodor

```
> rpcinfo -p ultra
program vers proto  port
100000  4  tcp   111  rpcbind
100000  4  udp   111  rpcbind
100232  10  udp  32777  sadmind
100083  1  tcp  32775  ttdbserverd
100221  1  tcp  32777  kcms_server
100068  5  udp  32778  cmsd
100229  1  tcp  32779  metad
100230  1  tcp  32781  metamhd
100242  1  tcp  32783  rpc.metamedd
100001  4  udp  32780  rstatd
100002  3  udp  32782  rusersd
100002  3  tcp  32785  rusersd
100008  1  udp  32784  walld
100012  1  udp  32786  sprayd
100011  1  udp  32788  rquotad
100024  1  udp  32790  status
100024  1  tcp  32787  status
100133  1  udp  32790  nsm_addrand
100133  1  tcp  32787  nsm_addrand
[ Dozens of lines cut for brevity ]
```

This example shows that hosts frequently offer many RPC services, which increases the probability that one is exploitable. You should also notice that most of the services are on strange high-numbered ports (which may change for any number of reasons) and split between UDP and TCP transport protocols.

Because the RPC information is so sensitive, many administrators try to obscure this information by blocking the portmapper port (111). Unfortunately, this does not close the hole. Nmap can determine all of the same info by directly communicating with open RPC ports through a 3-step process

1. The TCP and/or UDP port scan finds all of the open ports
2. Version detection determines which of the open ports use the SunRPC protocol
3. The RPC brute force engine determines the program identity of each rpc port by trying a "NULL command" against each of the 600 programs numbers in nmap-rpc. Most of the time Nmap guesses wrong and receives an error message stating that the requested program number is not listening on the port. Nmap continues trying each number in its list until success is returned for one of them. Nmap gives up in the unlikely event that it exhausts all of its known program numbers or if the port sends malformed responses that suggest it is not really RPC.

The RPC program identification probes are done in parallel, and retransmissions are handled for UDP ports. This feature is automatically activated whenever version detection finds any RPC ports. Or it can be performed without version detection by specifying the `-sR` option. Example 7, "[Nmap direct RPC scan](#)" demonstrates direct RPC scanning done as part of version detection.

## Example 7. Nmap direct RPC scan

```
# nmap -F -A -sSU ultra

Starting nmap 3.48 ( http://www.insecure.org/nmap/ )
Interesting ports on ultra.nmap.org (192.168.0.50):
```

# Nmap Service & Application Version Detection

By Fyodor

```
(The 2171 ports scanned but not shown below are in state: closed)
```

```
PORT  STATE SERVICE  VERSION
```

```
[A whole bunch of ports cut for brevity]
```

```
32776/tcp open  kcms_server  1 (rpc #100221)
```

```
32776/udp open  sadmin  10 (rpc #100232)
```

```
32777/tcp open  kcms_server  1 (rpc #100221)
```

```
32777/udp open  sadmin  10 (rpc #100232)
```

```
32778/tcp open  metad  1 (rpc #100229)
```

```
32778/udp open  cmsd  2-5 (rpc #100068)
```

```
32779/tcp open  metad  1 (rpc #100229)
```

```
32779/udp open  rstatd  2-4 (rpc #100001)
```

```
32780/tcp open  metamhd  1 (rpc #100230)
```

```
32780/udp open  rstatd  2-4 (rpc #100001)
```

```
32786/tcp open  status  1 (rpc #100024)
```

```
32786/udp open  sprayd  1 (rpc #100012)
```

```
32787/tcp open  status  1 (rpc #100024)
```

```
32787/udp open  rquotad  1 (rpc #100011)
```

```
Device type: general purpose
```

```
Running: Sun Solaris 9
```

```
OS details: Sun Solaris 9
```

```
Uptime 0.120 days (since Sun Nov 16 21:38:16 2003)
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 252.701 seconds
```

## SSL Post-processor notes

As discussed in the technique section, Nmap has the ability to detect the SSL encryption protocol and then launch an encrypted session through which it executes normal version detection. As with the RPC grinder discussed previously, the SSL postprocessor is automatically executed whenever an appropriate (SSL) port is detected. This is demonstrated by Example 8, "Version scanning through SSL".

## Example 8. Version scanning through SSL

```
nmap -P0 -sSV -T4 -F www.amazon.com
```

```
Starting nmap 3.48 ( http://www.insecure.org/nmap/ )
```

```
Interesting ports on 207-171-184-16.amazon.com (207.171.184.16):
```

```
(The 1214 ports scanned but not shown below are in state: filtered)
```

```
PORT  STATE SERVICE  VERSION
```

```
80/tcp open  http    Apache Stronghold httpd 2.4.2 (based on Apache 1.3.6)
```

```
443/tcp open  ssl/http Apache Stronghold httpd 2.4.2 (based on Apache 1.3.6)
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 35.038 seconds
```

Note that the version information is the same for each of the two open ports, but the service is http on port 80 and ssl/http on port 443. The common case of https on port 443 is not hardcoded - Nmap should be able to detect SSL on any port and determine the underlying protocol for any service that Nmap can detect in cleartext. If Nmap had not detected the server listening behind SSL, the service listed would be "ssl/unknown". If Nmap had not been built with SSL support, the service listed would have simply been "ssl". The "version" column would be blank in both of these cases.

# Nmap Service & Application Version Detection

By Fyodor

The SSL support for Nmap depends on the free OpenSSL library and has not been tested on Windows. Nor is it included in the Linux RPM binaries, to avoid breaking systems which lack these libraries. The Nmap source code distribution attempts to detect OpenSSL on a system and link to it when available. See chapter one for details on customizing the build process to include or exclude OpenSSL.

## nmap-service-probes File Format

As with remote OS detection (-O), Nmap uses a flat file to store the version detection probes and match strings. While the version of nmap-services distributed with Nmap is sufficient for most users, understanding the file format allows advanced Nmap hackers to add their own services to the detection engine. Like many UNIX files, nmap-service-probes is line-oriented. Lines starting with a hash (#) are treated as comments and ignored by the parser. Blank lines are ignored as well. Other lines must contain one of the directives described below. Some readers prefer to peek at the examples in the section called "Putting it all together" before tackling the following dissection.

### The Exclude directive

Syntax: Exclude <port specification>

Examples:

```
Exclude 53,T:9100,U-30000-40000
```

This directive excludes the specified ports from the version scan. It can only be used once and should be near the top of the file, above any probe directives. The Exclude directive uses the same format as the Nmap -p switch, so ranges and comma separated lists of ports are supported. In the nmap-service-probes included with Nmap the only port excluded is TCP 9100. This is a common port for printers to listen on and they often print whatever data is sent to them. So a version detection scan can cause them to print many pages full of probes that Nmap sends, such as SunRPC requests, Help statements, and X11 probes.

This behaviour is often undesirable, especially when a scan is meant to be stealthy. However, Nmap's default behaviour of avoiding scanning this port can make it easier for a sneaky user to hide a service: simply run it on port 9100 and it is less likely to be identified by name (the port scan will still show it as open). Users can override the Exclude directive with the --allports option. This causes version detection to interrogate all open ports.

### The Probe directive

Syntax: Probe <protocol> <probename> <probesendstring>

Examples:

```
Probe TCP GetRequest q|GET / HTTP/1.0\r\n\r\n|
Probe UDP DNSStatusRequest q|\0\0\x10\0\0\0\0\0\0\0\0|
Probe TCP NULL q||
```

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

# Nmap Service & Application Version Detection

By Fyodor

## protocol

This must be either TCP or UDP. Nmap only uses probes that match the protocol of the service it is trying to scan.

## probename

This is a plain English name for the probe. It is used in service fingerprints to describe which probes elicited responses.

## probestring

Tells Nmap what to send. It must start with a q, then a delimiter character which begins and ends the string. Between the delimiter characters is the string that is actually sent. It is formatted similarly to a C or Perl string in that it allows the following standard escape characters: \\ \0, \a, \b, \f, \n, \r, \t, \v, \xHH. One Probe line in nmap-service-probes has an empty probestring, as shown in the third example above. This is the TCP NULL probe which just listens for the initial banners that many services send.

## The match directive

Syntax: match <service> <pattern> [*versioninfo*]

Examples:

```
match ftp m/^220.*Welcome to PureFTPd (\d\S+)/ p/PureFTPd/ v/$1/
match ssh m/^SSH-(.[\d]+)-OpenSSH_(\S+)/ p/OpenSSH/ v/$2/ i/protocol $1/
match mysql m/^.\0\0\0\n(4\.[-\w]+)\0...\0/s p/MySQL/ i/$1/
match chargen m|@ABCDEFGHIJKLMNOPQRSTUVWXYZ|
match citrix-ica m|^x7fx7fICA\0x7fx7fICA\0| p/Citrix Metaframe XP ICA/ o/Windows/
match finger m|\r\n {4}Line {5,8}User {6,8}Host\s\ {13,18}Idle +Location\r\n| p/Cisco fingerd/ o/IOS/
d/router/
match netbios-ssn m+^.\0\0\0.\xffSMB\r0.*([\^0][([\^w\0]))([\-w\0]{2,50})+ p/Samba smbd/ v/3.X/
i/workgroup: $P(3)/
```

The match directive tells Nmap how to recognize services based on responses to the string sent by the previous Probe directive. A single Probe line may be followed by dozens or hundreds of match statements. If the given pattern matches, an optional version specifier builds the application name, version number, and additional info for Nmap to report. The arguments to this directive follow:

## service

This is simply the service name that the pattern matches. Examples would be ssh, smtp, http, or snmp.

## pattern

This pattern is used to determine whether the response received matches the service given in the previous parameter. The format is like Perl, with the syntax being "m/[regex]/[opts]". The "m" tells Nmap that a match string is beginning. The forward slash (/) is a delimiter, which can be substituted by almost any printable character as long as the second slash is also replaced

## Nmap Service & Application Version Detection

By Fyodor

to match. The regex is a Perl-style regular expression. This is made possible by the excellent Perl Compatible Regular Expressions (PCRE) library (<http://www.pcre.org>). The only options currently supported are 'i', which makes a match case-insensitive and 's' which includes newlines in the '.' specifier. As you might expect, these two options have the same semantics as in Perl. Subexpressions to be captured (such as version numbers) are surrounded by parenthesis as shown in most of the examples above.

### versioninfo

This field is of the form p/vendorproductname/ v/version/ i/info/ h/hostname/ o/operatingsystem/ d/devicetype/ where the slashes can be replaced by any delimiting characters. Any of the 6 fields can be omitted. In fact, all of the fields can be omitted if no further information on the service is available. The vendorproductname includes the vendor and often service name when relevant and is of the form "Sun Solaris rexecd", "ISC Bind named", or "Apache httpd". The version string is the version "number" (may include non-numeric characters, and even multiple words). "info" is miscellaneous further information that was immediately available and might be useful (like whether an X server is open, or the protocol number of ssh servers). "hostname" is filled out whenever a service offers up a hostname that the service uses. This is common for protocols like SMTP and POP3 and is useful because these hostnames may be for internal networks or otherwise differ from the straightforward reverse DNS response. "operatingsystem" and "devicetype" are present when the service's operating system and/or device type are known. These correspond to the descriptions offered up by Nmap's OS detection system (-O) and are intended to complement this feature. Any of the version fields can include numbered strings such as \$1 or \$2, which are replaced (in a Perl-like fashion) with the corresponding parenthesized substring in the *pattern*. In rare cases, a helper function can be applied to the replacement text before insertion. The \$P(3) expression in the example netbios-ssn match string above is one such example. The P() function includes only printable characters from the captured string. For netbios-ssn, a string such as "W\0O\0R\0K\0G\0R\0O\0U\0P\0" is decoded to simply "WORKGROUP".

### The softmatch directive

Syntax: `softmatch <service> <pattern>`

Examples:

```
softmatch ftp m/^220 [-.\w ]+ftp.*\r\n$/i
softmatch smtp m|^220 [-.\w ]+SMTP.*\r\n|
softmatch pop3 m/^\+OK [-\[\]\(\)!./+:<>@.\w ]+\r\n$/
```

The softmatch directive is similar in format to the match directive discussed above. The main difference is that scanning continues after a softmatch, but it is limited to probes that are known to match the given service. This allows for a normal ("hard") match to be found later, which may provide useful version information. See the section called "Technique Described" for more details on how this works. Arguments are not defined here because they are the same as for 'match' above, except that there is never a *versioninfo* argument. Also as with match, many softmatch statements can exist within a single Probe.

# Nmap Service & Application Version Detection

By Fyodor

## The ports and sslports directives

Syntax: ports <portlist>

Examples:

```
ports 21,43,110,113,199,505,540,1248,5432,30444  
ports 111,4045,32750-32810,38978
```

This line tells Nmap what ports the services identified by this Probe are commonly found on. It should only be used once within each Probe section. The syntax is a slightly simplified version of that taken by the Nmap -p option. See the examples above. More details on how this works are in the section called "Technique Described"

Syntax: sslports <portlist>

Example:

```
sslports 443
```

This is the same as 'ports' directive described above, except that these ports are often used to wrap a service in SSL. For example, the HTTP probe declares 'sslports 443' and SMTP-detecting probes have an 'sslports 465' line because those are the standard ports for https and smtps respectively. The *portlist* format is the same as with ports. This optional directive cannot appear more than once per Probe.

## The totalwaitms directive

Syntax: totalwaitms <milliseconds>

Example:

```
totalwaitms 5000
```

This rarely necessary directive specifies the amount of time Nmap should wait before giving up on the most recently defined Probe against a particular service. The Nmap default is usually fine.

## The rarity directive

Syntax: rarity <value between 1 and 9>

Example:

```
rarity 6
```

The rarity directive roughly corresponds to how frequently this probe can be expected to return useful results. The higher the number, the more rare the probe is considered and the less likely it is to be tried against a service. More details can be found in the section called "Probe Selection and Rarity".

# Nmap Service & Application Version Detection

By Fyodor

## The fallback directive

Syntax: fallback <Comma separated list of probes>

Example:

```
fallback GetRequest,GenericLines
```

This optional directive specifies which probes should be used as fallbacks for if there are no matches in the current Probe section. For more information on fallbacks see the section called “Cheats and Fallbacks”. For TCP probes without a fallback directive, Nmap first tries match lines in the probe itself and then does an implicit fallback to the NULL probe. If the fallback directive is present, Nmap first tries match lines from the probe itself, then those from the probes specified in the fallback directive (from left to right). Finally, Nmap will try the NULL probe. For UDP the behaviour is identical except that the NULL probe is never tried.

## Putting it all together

Here are some examples from nmap-service-probes which put this all together (to save space many lines have been skipped). After reading this far into the section, the following should be understood.

```
# The Exclude directive takes a comma separated list of ports.
# The format is exactly the same as the -p switch.
Exclude T:9100

# This is the NULL probe that just compares any banners given to us
#####NEXT PROBE#####
Probe TCP NULL q|
# Wait for at least 5 seconds for data. Otherwise an Nmap default is used.
totalwaitms 5000
# Windows 2003
match ftp m/^220[ -]Microsoft FTP Service\r\n/ p/Microsoft ftpd/
match ftp m/^220 ProFTPD (\d\S+) Server/ p/ProFTPD/ v/$1/
softmatch ftp m/^220 [-.\w ]+ftp.*\r\n$/i
match ident m|^flock\(\) on closed filehandle .*midentd| p/midentd/ i/broken/
match imap m|^* OK Welcome to Binc IMAP v(\d[-.\w]+)| p/Binc IMAPd/ v$1/
softmatch imap m/^* OK [-.\w ]+imap[-.\w ]+\r\n$/i
match lucent-fwadm m/^0001;2$/ p/Lucent Secure Management Server/
match meetingmaker m/^\xc1,$/ p/Meeting Maker calendaring/
# lopster 1.2.0.1 on Linux 1.1
match napster m/^\1$/ p/Lopster Napster P2P client/

Probe UDP Help q|help\r\n\r\n|
rarity 3
ports 7,13,37
match chargen m|@ABCDEFGHIJKLMNOPQRSTUVWXYZ|
match echo m|^help\r\n\r\n$|
# Will last until 0xC5FFFFFF, in April 2005 - need to shift in advance.
match time m/^\[xc0-\xc5]...$|
```

# Nmap Service & Application Version Detection

By Fyodor

## Community Contributions

No matter how technically advanced a service detection framework is, it would be nearly useless without a comprehensive database of services against which to match. This is where the open source nature of Nmap really shines. The Insecure.Org lab is pretty substantial by geek standards, but it can never hope to run more than a tiny percentage of machine types and services that are out there. Fortunately experience with OS detection fingerprints has shown that Nmap users together run all of the common stuff, plus a staggering array of bizarre equipment as well. The Nmap OS Fingerprint Database contains more than a thousand entries, including all sorts of switches, WAPs, VoIP phones, game consoles, UNIX boxes, Windows hosts, printers, routers, PDAs, firewalls, etc. Version detection also supports user submissions, and Nmap users have contributed thousands of services. There are three primary ways that the Nmap community helps to make this an exceptional database:

1. *Submit service fingerprints* -- If a service responds to one or more of Nmap's probes and yet Nmap is unable to identify that service, Nmap prints a "service fingerprint" like this one:

```
SF-Port21-TCP:V=3.40PVT16%D=9/6%Time=3F5A961C%r(NULL,3F,"220\x20stage\x20F
SF:TP\x20server\x20(Version\x202.1WU(1)\+SCO-2.6.1\+-sec)\x20ready\
SF:\r\n")%r(GenericLines,81,"220\x20stage\x20FTP\x20server\x20(Version\x
SF:202.1WU(1)\+SCO-2.6.1\+-sec)\x20ready.\r\n500\x20":\x20command\
SF:x20not\x20understood.\r\n500\x20":\x20command\x20not\x20understood.\
SF:r\n");
```

If you receive such a fingerprint, and are sure you know what daemon version is running on the target host, please submit the fingerprint at the URL Nmap gives you. The whole submission process is anonymous (unless you choose to provide identifying info) and should not take more than a couple minutes. If you are feeling particularly helpful, scan the system again using `-d` (Nmap sometimes gives longer fingerprints that way) and paste both fingerprints into the fingerprint box on the submission form. Sometimes people read the file format section and submit their own working match lines. This is OK, but please submit the service fingerprint(s) as well because existing scripts make integrating and testing them relatively easy.

For those who care, the information in the fingerprint above is port number (21), protocol (TCP), Nmap version (3.40PVT16), date (September 6), UNIX time in hex, and a sequence of probe responses in the form `r({probename}, {responselength}, "{responsestring}")`

2. *Submit corrections* -- This is another easy way to help improve the database. When integrating a service fingerprint submitted for "chargen on Windows XP" or "FooBar FTP server 3.9.213", it is difficult to determine how general the match is. Will it also match chargen on Solaris or FooBar FTP 2.7? There is no good way to tell. So a very specific name is used in the hope that people will report when the match needs to be generalized. If you scan a host and the service fingerprint gives an incorrect OS, version number, application name, or even service type, please mail the full Nmap output and correct information to `<fyodor@insecure.org>` and Nmap will be updated appropriately.
3. *Submit new probes* -- Suppose Nmap fails to detect a service. If it received a response to any probes at all, it should provide a fingerprint that can be submitted as described in #1 above. But what if there is no response and thus a fingerprint is not available? Create and submit your own probe! These are very welcome. The following steps describe the process.

## Steps for creating a new version detection probe

## Nmap Service & Application Version Detection

By Fyodor

- a. Download the latest version of Nmap from <http://www.insecure.org/nmap/> and try again. You would feel a bit silly spending time developing a new probe just to find out that it has already been added. Make sure no fingerprint is available, as it is better to recognize services using existing probes if possible than to create too many new ones. If the service does not respond to any of the existing probes, there is no other choice.
- b. Decide on a good probe string for recognizing the service. An ideal probe should elicit a response from as many instances of the service as possible, and ideally the responses should be unique enough to differentiate between them. This step is easiest if you understand the protocol very well, so consider reading the relevant RFCs and product documentation. One simple approach is to simply start a client for the given service and watch what initial handshaking is done by sniffing the network with Ethereal or Tcpdump, or connecting to a listening Netcat.
- c. Once you have decided on the proper string, add the appropriate new Probe line to Nmap (see the section called "Technique Described" and the section called "[nmap-service-probes](#) File Format"). Do not put in any match lines at first, although a 'ports' directive to make this new test go first against the registered ports is OK. Then scan the service with Nmap a few times. You should get a fingerprint back showing the service's response to your new probe. Send the new probe line and the fingerprints (against different machines if possible, but even a few against the same daemon helps to note differences) to Fyodor at [fyodor@insecure.org](mailto:fyodor@insecure.org). It will likely then be integrated into future versions of Nmap. Any details you can provide on the nature of your probe string is helpful as well. For custom services that only appear on your network, it is better to simply add them to your own nmap-service-probes rather than the global Nmap

**[RECIPE] Find all servers running an insecure or nonstandard version of an application**

*I haven't actually written this recipe yet. Sorry.*

**[RECIPE] Hack version detection to suit custom needs, such as open proxy detection**

*I haven't actually written this recipe yet. Sorry.*