Remote SMTP Server Detection

Julien Bordet <zejames@greyhats.org>

September 4, 2002

Abstract

While doing penetration tests, first one must retrieve as much information about remote server as he can. Some existing tools are very useful for that purpose, allowing for exemple to detect which Operating System is used thanks to network stack reaction. Some others operate at the application level : they detect remote software version through the way it responds to special requests.

This article introduces smtpscan, a new detection tool written to allow remote detection of SMTP servers.

1 Introduction

Since the introduction of Fyodor's NMAP[1] and its fingerprints [2], lots of technics to remotly gather information about remote servers have been used. Simple banners reading, TCP or ICMP answers analysis are very useful. Once OS version has been detected, one often wish to know which sofware version is used. Again, the easier and most common way to do it is to read the corresponding banner :

```
[julien@lysis smtpscan] nc mail.test.com 25
220 mail.test.com ESMTP Sendmail 8.10.1/8.10.1; Sun, 1 Sep 2002 14:30:02 +02
```

But this method is not very reliable : lots of remote servers allows administrators to configure and modify banners. Another way must be found.

In fact, the same method as nmap's fingerprinting can be used here : by sending requests that do not exactly correspond to protocols' standarts, by using weird or rare options, one must get a 'software fingerprint' that may help to identify which software is used. That is exactly what smtpscan does for SMTP servers.

2 Fingerprinting methodology

SMTP server behaviour is defined by several RFCs. The most relevant are RFCs 821 (*SIMPLE MAIL TRANSFER PROTOCOL*), 1425 (*SMTP Service Extension*) and 1985 (*SMTP Service Extension for Remote Message Queue Starting*). Standart states commands that can be used by a SMTP client, mandatory features that must be implemented by all SMTP servers, valid arguments and data that can be understood.

But as usual, implementation do not exactly correspond to what RFCs say. That little differences are used by smtpscan.

Of course, error messages could also be used (for exemple and Sendmail does not say 'Hello' as a Postfix does). But

- they can be personnalized on serveral servers, so they are much less reliable,
- using error code is enough :)

[1] http://www.insecure.org/nmap

[2] Remote OS detection via TCP/IP Stack FingerPrinting, http://www.insecure.org/nmap/nmap-fingerprinting-article.html

The tests currently used by smtpscan are the following :

- **Send a valid MAIL FROM without saying HELO first** : some servers allow clients to do it, and answer with a 220 error code, some other don't (and refuse it with a 501 or 503).
- **Send a HELO without a domain name after it** : from the RFCs, this is not mandatory, but some servers accepts it and some don't
- **Use a MAIL FROM test without a ':' between FROM and test** : this is explicitely required, but some MTA, like qmail, accept a MAIL FROM without ':'.
- **Use a MAIL FROM: <> with an empty from address** : any servers should accept this, but there are always exceptions
- **Use a MAIL FROM specifying the source address without closing bracket '>'** : this is normally not valid, but some servers accept it (because some clients may use it...)
- **Use a invalid source address** : Some servers check for a valid domain name in the source address
- **Use a simple 'test' recipient** : This normaly refers to a local user, but may not be allowed by some MTAs.
- **Use the HELP command** : may or may not be implemented
- **Use the VRFY command** : may or may not be implemented
- **Use the EXPN command** : may or may not be implemented
- **Use the TURN command** : may or may not be implemented
- **Use the SOML command** : may or may not be implemented
- **Use the SAML command** : may or may not be implemented
- **Use the NOOP command** : may or may not be implemented
- **Use the EHLO command** : may or may not be implemented

This set of tests proves to be a good way of guessing the remote server type.

3 smtpscan implementation

smtpscan is a Perl implementation of this method. It consists of approximatively 260 lines of code.

It is divided into three parts :

- the test file, that contains the commands that must be passed to the remote server. They are described above.
- the fingerprint file, that contains data about know SMTP servers.
- the script itself, called smtpscan, that loads tests from test file, and applies it to the remote server. Then it compares the fingerprints it got with the fingerprints contained in the database. If there is not exact match, it prints the nearest matches (counting the differences)

As usual with fingerprints, the database can be completed with untested servers, and may evolve in the futur. Moreover, the number of tests could easily increase if needed, if for instance there are too many 'collision' (same fingerprint with different servers). Indeed tests as well as fingerprints are completely dynamic and not hardcoded in the program.

The fingerprint database consists in one line per fingerprint. The first field is used to described remote server versions, and remaining fields correspond to the fingerprint itself, that it error codes answered to our test requests . For exemple :

```
Sendmail 8.12.2:250:501:501:250:553:553:550:214:250:250:502:502:502:250:250
Exim 3.12:250:501:500:250:501:250:501:214:250:550:500:500:500:250:250
```

So far the database contains 77 different fingerprints. But user report would be very helpful to increase it.

4 Conclusion

The smtpscan implementation for the method presented here has proven so far to be vey efficient. Indeed, no 'collision' has yet been found : different servers have always given different fingerprints (that is, of course, not true for subversions, that can however often be determined). Consequently, it may really useful while doing penetration tests.

It may be noticed that, of course, every 'rich' (that is with numerous text commands and numerous specifications) existing or futur protocol could be applied the same method, and maybe with a few modification with smtpscan itself.

5 Greetings

- Fyodor for its great nmap tool, and its fingerprinting implementation
- Jedi/Sector One for its tool Ftpmap, that has given my the idea of writting smtpscan
- Tomat (tomat@bigbuffer.org) for re-reading this document and his precious advices
- Gwen forever