Vulnerability Testing Against a Predominately UNIX-based Network (DRAFT)

Jeffrey S. Marker, CISSP

April 22, 2003

1 Introduction

Network vulnerability testing seems to be a self-describing term. When we perform this type of testing, we are interested in where and how hosts may be attacked via the network. We are concerned about potential vulnerabilities, not with exploiting these vulnerabilities. Exploiting vulnerabilities to demonstrate their danger is the providance of *penetration testing*, where the goal is to demonstrate how a networked host can be compromised.

We are limiting our discussion to "predominately UNIX" networks because we wish to address TCP/IP testing. We do not want to discuss NetBIOS or other non-TCP/IP protocols. The general methodology described herein should be applicable to non-TCP/IP protocols, although the tools may be different.

Network vulnerability testing is valuable for a number of reasons. The initial tests will help the systems, network, and security administrators develop a picture of what is present on the network. Subsequent tests can help map changes to the network. Regular testing can help discourage the running of unauthorized services. Vulnerability testing also helps demonstrate the effectiveness of current security measures, and the reports from the tests can be used to help evaluate, and make a case for, enhanced security measures. And, finally, there is always the remote chance that one might detect compromised hosts on the tested network[1]

Vulnerability testing should *not* be performed to

[1]Vulnerability testing should not be thought of as a substitute for intrusion detection technologies.

1. Find hosts on the network to compromise,

2. Try to demonstrate that Biff down the hall does not know how to secure a computer.

The first of these reasons is of questionable legal and ethical standing, while the latter is simply juvenile.

2 Preparing for the testing

Network connectivity

To perform the vulnerability testing, we need to have at least one network connection. This can be either "internal" – inside the security perimeter – or "external." Ideally, we would be able to have both an internal and an external connection, so we can compare the views of an external and internal threat.

External network connection

Simply put, the purpose of the external network connection is to allow us to see what an "outsider" will see during an attack. This connection should be outside of whatever perimeter security is in place for the network to be tested. Also, all automatic trust relationships between the testing platform and the hosts to be tested should be severed for the duration of the testing. This means that SSH keys, *.rhosts*, */etc/hosts*, *.netrc* files and the like need to be checked to make certain that they are not allowing the testing platform access that other hosts or users can not be expected to have. Failure to sever these trust relationships may ultimately result in false-positive reports, which will make it appear that the network being tested is less secure than it actually is.

Because remote network vulnerability testing can be rather network intensive, it is best if the external network connection has a lot of bandwidth. This can be accomplished via the sundry commercial broadband solutions[2], by locating the testing platform at the site of a business partner, or by locating the testing platform at a remote site.

[2]such as DSL or cable modems.

Internal network connection

The internal network connection is used for the internal testing. This allows us to see what "insiders" see, and to be able to make some value judgements regarding the effectiveness of the perimeter security. This connection should be inside the security perimeter of the network we are testing. Unlike the external testing, we do not need to sever our trust relationships, because it can be assumed that trust exists with other hosts – and other users – on the network.

Testing platform

The tools we will use to perform our testing are all UNIX based. Consequently, the testing platform will need to be able to run a version of UNIX as its operating system.

Tools

nmap

nmap is primarily a port scanner, rather than a vulnerability tester per se. We will use this tool in both the external and internal testing to determin which TCP and UDP services are listening on the tested hosts. We will also make use of *nmap*'s operating system detection feature to make guesses at what operating systems are being run on the tested network. Finally, we will use some of *nmap*'s scanning variations in the external testing to map out the perimeter security.

nmap can be obtained from *http://www.insecure.org/nmap/*.

SATAN

SATAN is the grand old man of network vulnerability testers, being released in 1995. Because of its age, it is often called obsolete. However, *SATAN* uses it's own method for probing services, which gives us another view of the network. Additionally, it was written with modularity in mind, so enhancing *SATAN* with localized tests is well-documented[3] and relatively straightforward.

SATAN can be obtained from *http://www.porcupine.org/satan/*.

[3] See, for example, *Protecting Networks with SATAN* by Martin Freiss.

SARA

SARA is a third-generation decendant of *SATAN*, with all of the advancements one might expect to find. It has hooks for using *nmap* for service scanning, an up-to-date vulnerability database, and a well-developed report writing feature.

SARA can be obtained from *http://www-arc.com/sara/*.

Nessus

Nessus is another modern vulnerability scanner. Unlike the *SATAN* family, it uses it's own console for configuration and reporting[4], and is client-server based. Also, *Nessus* makes it easy to select precisely which tests to run, whereas the *SATAN* family selects its tests based upon a "scanning level."

Nessus can be obtained from *http://www.nessus.org/*.

Whisker

Whisker tests http servers for vulnerabilities. It will, therefore, only be used if hosts on the network are found to be running http servers.[5]

Whisker can be obtained from *http://sourceforge.net/projects/whisker/*.

House-keeping

Make a directory for the results of the tests. For the examples herein, we'll call that *example.com_vuln_test*. Under that directory, we will make *external* and *internal*, so as to be able to better organize the different tests.

Protection

When we perform the tests, there are two items we need to protect: ourselves and the data the testing generates. We need to protect ourselves from "real world" ramifications, such as letters of reprimand, demotion, firing, and/or various legal actions. We need to protect both the confidentiality and the integrity of the data.

[4]The *SATAN* family uses a Web browser.

[5]Note that this is not the same thing as "hosts listening on TCP port 80.

We protect ourselves by obtaining approval to perform the testing *before* we begin. In some organizations, this includes obtaining permission to possess the above mentioned tools[6]. Regardless of the organization, however, pre-approval from the *network owner* must be obtained. This means getting approval from management, not the network administrators. We want to get the approval from as far up the food chain as is possible.

We protect the data via file permissions, encryption, and labeling. Our file permissions should be as restrictive as is possible, allowing only the owner access[7]. We should also encrypt the data while it is stored[8]. Additionally, if the final report is to be transferred electronicly, it should be done so in an encrypted form[9]. Finally, the report should be properly labeled[10]. This helps prevent accidental disclosure via honest mistake, as well as encouraging the authorized recipients to be more protective of the document.

3 Testing

One important rule is to not launch denial of service attacks against production networks during "normal" production hours[11]. Some of the tests the various tools perform have the potential to cause certain network services to "go away." Also, some of the tools have to potential to perform a netwrk bandwidth denial of service, either intentionally via certain tests or unintentionally because too many tests are being run at once.

[6]Some organizations have strict policies regulating which job functions are allowed to possess "cracker" tools.

[7]It is possible that the group might be allowed access, if the testing and analysis is being performed by a team.

[8]The discussion of cryptographic tools is beyond the scope of this document. However, we have used both Matt Blaze's *cfs* (http://www.crypto.com/papers/cfs.pdf) and *gnupg* (http://www.gnupg.org/) with success.

[9]Methods of encrypting electronic mail are many – one might start with S/MIME or *gnupg*. Methods of encrypting data sent to network printers are not as well documented.

[10]ie. "Confidential: disclose and distribute to Jiffy Script, Inc. employees having a need to know," or "Ultra secret."

[11]The windows when hosts or networks can be taken down for maintenance will be defined in the organization's policies.

3.1 External testing

If perimeter security is in force on the network to be tested, it is possible that we are able to know what security policies are being enforced on the perimeter. This "insider" knowledge will modify the *nmap* portion of the test, as we can avoid a comprehensive port scan, and, instead, scan only those services which are allowed to pass through the perimeter.

Open service detection with *nmap*

We begin the testing with *nmap*, because that will give us a picture of open services. First, we will will probe all TCP ports, via

```
% nmap -sT -p 1-65535 -P0 -n -oA tcp_services -v \
-iL hosts_to_scan
```

This will take quite some time to run, especially if the perimeter security utilizes rule that drop, rather than reject, unwanted packets. When it finishes, however, we will have three files called *tcp_services.gnmap*, *tcp_services.nmap*, and *tcp_services.xml* in the *example.com_vuln_test/external* directory.

Next, we will probe all UDP ports, via

```
% nmap -sU -p 1-65535 -P0 -n -oA udp_services -v \
-iL hosts_to_scan
```

As is the case with the TCP probe, this test will take quite a while to run. When is is finished, we will have files called *udp_services.gnmap*, *udp_services.nmap*, and *udp_services.xml* in the *example.com_vuln_test/external* directory.

Statistics, Research, and Test Creation

The next step is to create a frequency list of open services. This can be accomplished by using the *Perl*-script *service_freq.pl* (Figure 2) in the following fashion[12]:

```
% service_freq.pl tcp_services.xml | sort -rn +1
```

[12] There are, of course, other methods of accomplishing this.

This will provide a list of TCP services, sorted in reverse numerical order by the number of times they appear on the network. A sample of such a list can be found in figure 1 on page 7. Running

```
% service_freq.pl udp_services.xml | sort -rn +1
```

will produce a similar list of UDP services.

```
tcp-22: 30
tcp-443: 2
tcp-80: 2
tcp-53: 2
tcp-25: 2
tcp-6715: 1
tcp-6667: 1
tcp-31337: 1
tcp-23: 1
```

Figure 1: Sample TCP service frequency list

We examine the frequency lists for services we know *should* be running – such as tcp-22[13] and tcp-80[14] in the sample[15] – and for services we know should *not* be running – such as tcp-6667[16] in the sample. In the process, we will probably find services that are listed neither in */etc/services* or in the *tcp_services.nmap* or *udp_services.nmap* files. These should be researched[17], as should any named services that are unfamiliar or unexpected.

The vulnerability testing programs we are using will likely have tests for most, if not all, of the services *nmap* shows us. It is important to remember, however, that information security is a moving target. The tools we use can only find vulnerabilities for which they have tests, and vulnerabilites, as well as their corresponding exploits, are ever changing. Thus, it is possible, if not likely, that some services *nmap* tells us are running will not have tests

[13] SSH.
[14] HTTP.
[15] Figure 1 on page 7.
[16] IRC.
[17] See appendix B for hints on how to research mysterious services.

included in the standard package. Consequently, we may need to search the Internet for tests, or we may have to write our own[18].

Once we have tests to check for vulnerabilities on all of the services reported running, we're ready to begin the vulnerability testing proper.

Each of the tools we are using can be run from either the GUI or from the command-line in a "batch" mode. We use the latter because because we feel that doing so leaves more resources available for the tools, and because doing so makes running the tools from a shell-script an intuitive next step.[19]

It it tempting to run the tools in parallel, so as to complete the initial testing phase as quickly as is possible. This temptation must be resisted, lest the testing become a DOS attack against either the network being tested or the network performing the testing. This will mean that the testing will take longer to be performed.

3.2 Being tricky

If we are not privy to the perimeter security rules, we can make use of *nmap*'s built-in tricks to test whether the perimeter security is a smoke-screen.

3.3 Internal testing

4 Analyze

4.1 False positives

Many of the tools only check banners – they don't actually test the vulnerability. Because many network daemons allow the administrator to use arbitrary headers, it may be necessary to verify the results of such tools by hand, either exhaustively or by examining a random sample.

4.2 False negatives

The tools we use are only as knowledgeable as their vulnerability databases are up-to-date. Fortunately, most of these tools allow direct review of *all* of

[18] This is left as an exercise to the reader, although each of the tools does contain some advice on how to write tests. That's why we use open source.

[19] Running the tools can take a long time, and sitting and watching a status bar is less than stimulating.

the responses to their tests, without the filter of the front-end.

4.3 Comparison between external and internal tests

4.4 Danger signs

5 Conclusions

A service_freq.pl

```
#!/usr/bin/perl

## read nmap XML files and produce a list of services
## and their frequency

while (<>) {
# change "open" to "filtered" or "closed" if you are
# interested in those counts
    if (/port protocol=\"(\w+)\" portid=\"(\d+)\"><state state=\"open\"/) {
        $sc{"$1-$2"}++;
    }
}

foreach (keys %sc) {
    print "$_: $sc{$_}\n";
}
```

Figure 2: service_freq.pl

B Researching mysterious services

There are numerous lists of services common to various TCP ad UDP ports. Some are more complete than others, some focus entirely on what "trojan horses" run on which ports; some are updated regularly; some have remained

static for years. As with most information sources on the Internets, your milage may vary.

One relatively complete listing is the searchable "port report" at DShield.org, which really takes its descriptions from Neohapsis and the CVE. Because the list is searchable, one can quickly find the the information regarding the port. For example, the TCP service 31337 in the sample frequency list probably has no listing in */etc/services*, so we would go to www.dshield.org and query that service by point our browser at

http://www.dshield.org/port_report.php?port=31337

C What is not being tested

We have been describing network vulnerability testing, not penetration testing or the art of performing a security audit. It is important to note the general areas we do not test, if only to remind readers that the journey of security is never complete.

C.1 The system level

This method of testing does not examine system-level security issues. We do not examined the systems for weak static passwords[20]. We do not examined file-system permissions. We do not examine `/etc/hosts` or sundry `.rhosts` files for sanity, except for the "trust" tests performed for the **root** and/or **bin** users. We do not examine the *sudo(8)* configuration file for sanity. And, finally, we do not examine setuid permission, program versions, shared-library versions, or operating system patch levels, except as they may be revealed by network daemons.

Please note again that the list of system-level issues given in the previous paragraph is not inclusive, but is, instead, intended to provide a starting point, as well as to stress the complexity of system-level security.

[20] It can be argued that all static passwords are weak in the modern age.

C.2 Physical security

Physical security is often over-looked when an organization examines its security stance. The reasons for this are many, but included in them is the misbegotten believe that information security is only about firewalls, virus scanning, and intrusion detection — in other words, that information security is entirely in the "virtual" world.

C.3 Clandestine gathering

C.4 Social engineering

C.5 Source code reviews