

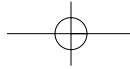
Chapter 10

Enterprise Scanning

Solutions in this Chapter:

- Planning a Deployment
- Configuring Scanners
- Data Correlation
- Common Problems

- ☑ Summary
- ☑ Solutions Fast Track
- ☑ Frequently Asked Questions



Introduction

Enterprise vulnerability scanning is quite complicated, and as such requires a certain amount of planning, preparation, and adjustment. The key factors for effectively scanning the enterprise for security vulnerabilities are easy administration, periodic scanning, and accurate results.

There is no trivial way to take a scanner such as Nessus and use it to scan the entire enterprise network. Simply pointing it toward the network and scanning will not be enough. This chapter shows some of the caveats that make this process difficult. You'll learn, for example, why simply scanning the entire network from a single point is often not viable. This involves exploring distributed scanning, differential reporting, report correlation, and automated updating.

At this point in the book, we expect that you are most likely already using Nessus for regular security testing, and are looking to take it up a notch—from maintaining a list of hosts you regularly scan, to scanning your entire enterprise and using the results to improve your enterprise's security status.

Planning a Deployment

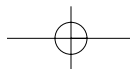
In the following section we will help you outline your plan for deployment.

Define Your Needs

Before scanning your enterprise network for security holes, you must remember that simply scanning anything that has an IP address will not bring you the expected benefits unless you can handle the huge number of vulnerabilities that are likely to appear in the report.

Our experience shows that on a typical vulnerability scan in a medium-to-large enterprise, each host scanned returns an average of 3 high-risk, 5 medium, and about 10 low-risk vulnerabilities. Quick math will show that scanning a small subnet of 100 hosts will return around 300 high-risk vulnerabilities and about 1,800 vulnerabilities in total. This computation doesn't take into account that some vulnerabilities might be the same on different computers, or that the same vulnerability might exist on different ports on the same machine.

According to an old Chinese proverb, “if *what we know* is the contour of a circle, and *what we do not know* is the inside of that circle, the *more we know* the *less we know that we do not know*.” We therefore must prepare beforehand so we're not overwhelmed by the amount of information we will receive once we start scan-



ning the network. We will divide the preparation into three parts: planning, preparation, and segmentation.

Planning

Some companies consider their customer database their most critical asset. Others consider the CEO's laptop most sensitive. Others still will mark their file server as the important one. Just as this is different for each company, each company's security needs are different, and so is the understanding of what "enterprise scanning" requires.

The easiest way to identify your most critical assets is to answer the following question: How much money and time will you lose if "something bad" happens to that asset? The definition of "something bad" is your worst-case scenario; this might be the deletion of a critical file or perhaps your database falling into the hands of your competition. Once you answer that question, you will see that scanning anything and everything doesn't necessarily solve the problem of scanning your most critical assets, as too much information is just as bad as too little. If you fail to notice vulnerabilities in your most critical assets because the report overflows with data about your least important hosts, you've simply missed important vulnerabilities for a pitiful reason.

When customers say they are uncertain of what is critical and what is not, it is good practice to revert to this list of items, ordered with more critical resources first:

1. Centralized servers (DNS, mail, file, database)
2. Financial servers and workstations
3. Management servers and workstations
4. Servers and workstations containing marketing data and plans
5. Sales servers and workstations
6. All remaining hosts

This list is a good starting point in identifying what the company management should worry most about an attacker compromising.

Scanning the right assets in the right way (most important first, or, alternatively, the most critical more frequently) will help you understand what is required to protect your critical machines.

This will prevent you from being overwhelmed by information and subsequently giving up the idea of scanning your enterprise. We have seen many cases

298 Chapter 10 • Enterprise Scanning

of companies that reached the unfortunate conclusion that vulnerability scanning is “impractical,” just because they were scanning the wrong parts of the network or just too much of it. This made it impossible for them to realize the obvious benefits if the vulnerability scanner was used properly. They threw away a tool that would be perfectly useful if it was simply tuned to only examine the most critical systems.

New security holes surface daily, and in order to be effective, you should plan to run your scans on a regular basis. It would be impractical to try to handle dozens and sometimes hundreds of reports illustrating the security vulnerabilities of the enterprise network without first prioritizing which reports are most important.

By the end of this section, you will be able to develop a list of critical assets (identified by hostnames, IP addresses, or even IP ranges) ordered by how important they are to your organization. Next to each asset, list what their acceptable “bill of health” is. You might decide that some servers must be completely free of vulnerabilities, while others might have medium- and low-risk vulnerabilities only. At the very least, do not accept any “high-risk” vulnerabilities on any of the critical assets. In addition, write down a point of contact for each asset, as this will be required once you start generating reports and need someone to address the issues found, and the frequency with which you will scan the asset. Table 10.1 shows one way of organizing the list of assets.

Table 10.1 Asset List

Asset Name	IP Ranges/ Hostnames	Expected Results	Manager Name and E-Mail	Frequency of Scans
HR database	192.168.1.50–250	No open ports No vulnerabilities	Joe Smith joes@acmecorp.net	Daily
R&D file servers	192.168.1.3–10	No open ports except file sharing and HTTP No high or medium vulnerabilities	John Williams johnw@acmecorp.net	Every other week

When it comes to the frequency of scans, we prefer *more* versus *less*. By *more*, we mean that running the scan once a day is preferable to running it once a month. Although this might sound extreme (we can imagine you’re now

thinking, “Do I really need to scan my network on a daily basis?”), you need to remember the recent Sasser worm. The Sasser worm hit the Internet just three weeks after the vulnerability was discovered. This means that a monthly vulnerability scan could have missed this vulnerability altogether, and as a result you would have missed the need to install this critical patch. This vulnerability discovery to worm creation window has actually been steadily shrinking. We cannot stress enough how important the frequency of your scans is to their success in preventing worm outbreaks at your site.

NOTE

Here are some vulnerability statistics from ICAT, available at <http://icat.nist.gov/icat.cfm?function=statistics>. In the first five months of 2004, 313 new vulnerabilities were discovered. In the year 2003, the total count was 1,007, and in the year 2002, the total number of vulnerabilities discovered was 1,308. Year 2001 holds the record for the last four years with 1,506.

The CERT reports, available at www.cert.org/stats/, show even higher figures. The number of vulnerabilities reported to CERT in the year 2003 was 3,784; in 2002, the number was 4,129; and in 2001, the count was 2,437.

According to SecuriTeam.com, an online vulnerabilities database, an average of about five new vulnerabilities are discovered each day. The total number of new vulnerabilities reported on the web site during the first five months of the year 2004 was over 750.

Most network administrators and security managers are extremely busy people who will find it difficult to handle a dozen or more scanning results a day, especially when they need to give these results the required attention. We will therefore show you in later sections ways to reduce the amount of reports generated to just one or two. Each of these reports will only show the changes since the last scan (if any) by using differential report generation.

Preparation

In many organizations, different managers are in charge of different assets. Each of these managers needs to be prepared for the vulnerability scans on his or her assets. In fact, in most cases, the scans will find vulnerabilities in the assets for

300 Chapter 10 • Enterprise Scanning

which the managers are responsible, so it's a good idea to prepare them for the potentially bad news. Furthermore, vulnerability scans are very “noisy”—the relevant network and system administrators need to be prepared to see abnormal log entries, higher CPU and memory consumption, and an increase in network traffic during the scanning phase.

Notes from the Underground....**The Importance of Prior Notification**

One of the authors was once present during a vulnerability scan that was done without prior notification to some of the system and database administrators. During the scans, some of the Solaris systems and the Oracle databases exhibited strange and abnormal log entries, all caused by the fact that the vulnerability scanner was sending unusual requests as part of the vulnerability assessment. Not knowing that a scan was in progress, the administrators immediately assumed the database was failing and started to run corrective measures.

You might be able to guess what happened next. The corrective actions, which were trying to fix a nonexistent problem, corrupted the database and destroyed the information stored on that machine. Hours of labor could have been saved if the administrators had been notified in advance of the pending scan and that unusual events would potentially occur during that time.

A good way to make sure everyone is aware that you are implementing enterprise-scaled vulnerability assessment is to invite the different managers to a meeting where you present the following:

- An overview of Nessus' capabilities.
- Different aspects of Nessus' effects.
- Live scans of a test environment, preferably a machine or subnet of low importance. This will better illustrate how the scan affects the machines and demonstrate the scanning process.

This last point, the demonstration of a live scan, should probably involve a look at the log messages generated on the target machines and might include a look at the network traffic through a sniffer like tcpdump or Ethereal.

By the end of this section, all of your asset managers should be aware of your intention to regularly scan them. They should understand that these scans will result in vulnerabilities being highlighted, which will then need to be addressed.

Finally, remember the politics of the situation. If you want a manager to be most responsive to vulnerability data, or any data presented by the security team, it's important to work hard to avoid an adversarial relationship. Approach the manager with a helpful tone and do what you can to make the process easier. Otherwise, you might find that either vulnerability scanning gets shut down or the results ignored. You can often help managers most by giving them a preliminary report, allowing them the chance to fix vulnerabilities and get a second scan in one to three day's time. This helps managers show that they are responsive and that vulnerabilities get a short lifespan on their watch.

Segmentation

As discussed earlier, your network contains different kinds of assets, each of which might have a different type of confidential material on it. The scan results, and scanning in general, might reveal sensitive information, such as usernames, weak passwords, hidden directories, and, of course, security vulnerabilities in those assets. To minimize information leakage, you should consider segmenting your scans, breaking them up so no one report contains information about the entire enterprise.

Segmentation allows you to test each of your assets while not providing confidential information about one set of assets to a manager of another set. In computer security, we're always thinking about risk avoidance and risk mitigation. In this situation, we're trying to decrease both the probability that sensitive information leaks and how much information a single leak can carry. Giving information about one asset's vulnerability to eight people instead of two roughly quadruples the risk. Remember, that person might turn against the organization, potentially criminally. He might drop the report on the ground inadvertently; he might get mugged. It's far easier to deal with these events if the report that's used maliciously contains less information.

There's more than just vulnerability information at stake, though. Nessus can be configured with or discover sensitive information, including logon information for Windows, brute-forcing results, SNMP community names, and so forth,

302 Chapter 10 • Enterprise Scanning

and as this type of information can show up in the vulnerability reports, you should segment the results in such a way that only the people who need access to each piece of information actually receive it. Another benefit of segmentation is that it allows you to use load-balancing techniques to optimize the scan based on your network topology and host concentrations.

Network Topology

Your organization's network topology largely affects the quality of the results you receive. For example, results from scanning the internal network (or MZ network) from your DMZ network will differ from a scan done from your internal network. Each of these scans is equally important, and the different results provide insight into your network's security situation. The difference is mainly caused by the effect that firewalls and other network devices have on the scan, as they allow or block connections, route or reroute traffic, according to a predefined rule set.

Therefore, a critical question to answer during the planning phase is, "what do you want to find?" Are you trying to see what a normal employee would be able to do to your file server, or what a night cleaning crewmember could do without having a valid username and password combination? Are you trying to find out what an Internet attacker would be able to do to your web server, or what a hacker could do if he successfully compromises the DMZ?

Each of these questions is equally important and requires a different type of deployment.

NOTE

At first glance, it might seem logical to install a scanning server on each of your different networks (DMZ, MZ, privileged network, etc.) and run VA scans from each to the rest of the organization. However, that would probably be a waste of time, as high-risk vulnerability on a sensitive host, no matter who has access to that network, should always be considered critical. Hiding a vulnerability doesn't fix it—it will eventually resurface. Network segmentation is an added protection, but should not be an alternative to fixing vulnerabilities.

The actual benefit of scanning the network from different points of view is in providing you with a feel of what needs to be addressed immediately and what can wait. For example, a high-risk vulnerability on your web server connected to the Internet needs greater attention than a high-risk file-sharing related vulnerability on the same web server does, as no outsider should have access to the related file-sharing port.

When segmentation is involved, we always recommend separating your network topology into two parts—internal and external—before dividing it further. “Internal” refers to the networks that are accessible only to employees, while “external” consists of all the hosts that are accessible from the Internet.

Your external topology should be scanned from an external server, imitating as much as possible an external attacker. The internal topology should be scanned from an internal server with full network access, imitating to the greatest extent possible an internal attacker.

NOTE

You should remember that when you receive the results for the external scans, every vulnerability should be accounted for, as it has been proven in the past that something considered a medium-risk vulnerability can quickly develop into a high-risk problem. In addition, low-risk vulnerabilities sometimes indicate that something is wrong: unnecessary services are installed on a hardened server, or incorrect rules are configured on the firewall. For example, your scan might find an unnecessary portmap or rpcbind process on a UNIX box and rank that as a low priority. A week later, when researchers release a remote-root vulnerability in Sun’s rpcbind, the risk due to that unnecessary service is much higher. It’s better to address even low-priority issues early, before the vulnerability escalates or a new one is discovered.

Bandwidth Requirements

One of the more important aspects of enterprise scanning is that unlike a one-time penetration test, typically done quarterly or even annually, enterprise scanning is done on a daily, weekly, or monthly basis. As such, network effects of the scan, such as bandwidth utilization or intrusiveness of the scans, are especially important.

304 Chapter 10 • Enterprise Scanning

Unfortunately, no network can provide us with unlimited bandwidth, and no two points on the network can use the complete bandwidth allowed by the network hardware without affecting other points on the network. Therefore, you must take into consideration that scanning your network will affect your network's overall performance. To complicate things even further, interoffice communications infrastructure tends to be even more restricted, and instead of 10/100/1000Mb per second, we can expect those communication lines to be 1Mb per second or less. In any case, this bandwidth needs to be shared with our coworkers—saturating the connection with our vulnerability scan will result in inaccurate scan reports due to packet loss, and lost of connectivity and functionality to our coworkers. Although Nessus does not consume much bandwidth if properly configured, scanning across a low-bandwidth and high-latency connection is not recommended unless there are no other alternatives.

How can we handle the problem of scanning multiple physical locations? One easy solution is to place a server in each location to make sure the available bandwidth between the scanning server and the network being scanned is at least 10Mb.

However, a better solution would be to first understand how much bandwidth Nessus requires, and enumerate any locations that do not meet these requirements. Before we lay out the bandwidth requirements, let's look at what affects bandwidth consumption. One simple rule of thumb is that a host with no open services requires far less bandwidth to scan than a server hosting multiple web servers, each running on a different port. The fewer services a host has, the fewer plugins Nessus will need to launch against it; thus, the lower bandwidth the Nessus scans will consume.

We have used freely available open-source tools to verify the bandwidth requirements needed for an average host. The tools are very easy to obtain, and require no knowledge of programming and very little technical skills. First, we need to use the ever-popular packet-capturing tool `tcpdump` (available from www.tcpdump.org). You can use simple capturing filters to capture only those packets originating from the scanning computer to the scanned host. We also use further filtering of the captured packets to prevent the capture file from becoming too large and difficult to handle.

For example, testing a port-80 scan using the filter `host 192.168.1.243 and port 80` allows the capture of all traffic originating from and destined for port 80. After capturing the entire Nessus session, you can generate statistics on the captured traffic. To do so, open the capture file using Ethereal (available from

www.ethereal.org) and choose **Statistics | Summary** (see Figure 10.1). Alternatively, if you want to create more in-depth graphs and statistics, you can use the tool `tcpstat` (available at www.frenchfries.net/paul/tcpstat/) with `gnuplot` (www.gnuplot.info). `Tcpstat` is able to take raw `tcpdump` files and generate numerical data from them. This numerical data combined with `gnuplot` allows you to take any numbers and crunch them into a graph.

We used the following steps to capture packets, take the capture file and convert it to numerical data, and generate from it an easy-to-analyze graph. We start with our packet-capturing command:

```
tcpdump -w iis.dump "host 192.168.1.243 and port 80"
```

Once a substantial amount of data is captured, we convert it to statistical information using:

```
tcpstat -r iis.dump -o "%R\t%B\n" 1 > iis.total.data \  
tcpstat -r iis.dump -f "dstport 80" -o "%R\t%B\n" 1 > iis.up.data \  
tcpstat -r iis.dump -f "srcport 80" -o "%R\t%B\n" 1 > iis.down.data
```

We then generate the corresponding graph by opening `gnuplot` and typing the following at the prompt:

```
set term png small  
set data style lines  
set grid  
set yrange [ -10 : ]  
set title "IIS Bandwidth"  
set xlabel "seconds"  
set ylabel "KBytes/s"  
plot "iis.total.data" using 1:($2/1024) smooth csplines title "Total"\  
    , "iis.up.data" using 1:($2/1024) smooth csplines title "Up"\  
    , "iis.down.data" using 1:($2/1024) smooth csplines title "Down"
```

The result of `gnuplot` will appear on the screen, and can be redirected to a file. For example, writing these lines to a file called `gnuplot.script` and running **`gnuplot gnuplot.script > picture.png`** will generate a file called `picture.png` containing the desired graph.

Now that we know how to generate the information, let's understand the different phases of Nessus scans and how much bandwidth they generate using the method just illustrated.

Portscanning Phase

During the portscanning phase, Nessus can use Nmap or a similar portscanner to determine what ports are open on a particular host. The Nmap “connect scan,” which uses a single TCP 72-byte packet with its SYN flag set, is Nessus’ default method to detect open ports. This packet is sent to each of the ports you configure Nessus to scan. Nessus by default will scan all ports between 1 and 15,000.

The number of packets sent depends on the latency of the scanning host, the network, and the host being scanned. On a very low latency and high-bandwidth network, this portscan can take roughly 1.5 seconds.

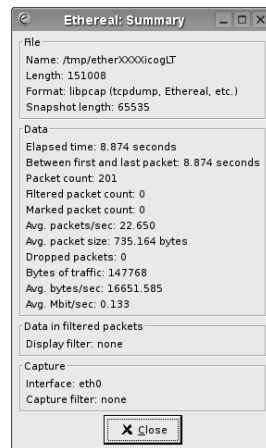
Doing some quick math brings us to:

$$15,000 \times 72 = 1,080,000 \text{ bytes}$$

The response packets require roughly the same amount of bandwidth. Dividing that by 15 seconds brings us to 720,000 bytes per second, or 720KBps. Although this isn’t very high, multiplying this by a few dozen hosts will generate enough traffic to temporarily bring a 100Mbit (roughly 12.5Mbytes per second) network to its knees. In addition, this large amount of packets will strain the firewalls and network devices responsible for connecting your scanning host with the host being scanned.

Moreover, not all traffic is created equal. We see in Figures 10.2 and 10.3 that the portscan phase generates average traffic—this average is controlled by how long we wait for a response from the host being portscanned. We will use this waiting time to reduce the bandwidth requirements, by setting the value to 1 millisecond (with the value of *Maximum wait between probes (ms)*). In addition, notice that there is a minimal difference between the bandwidth consumption of a TCP Connect() scan and a SYN scan.

Figure 10.1 Ethereal Summary



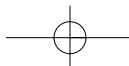


Figure 10.2 Portscan (1-15000)

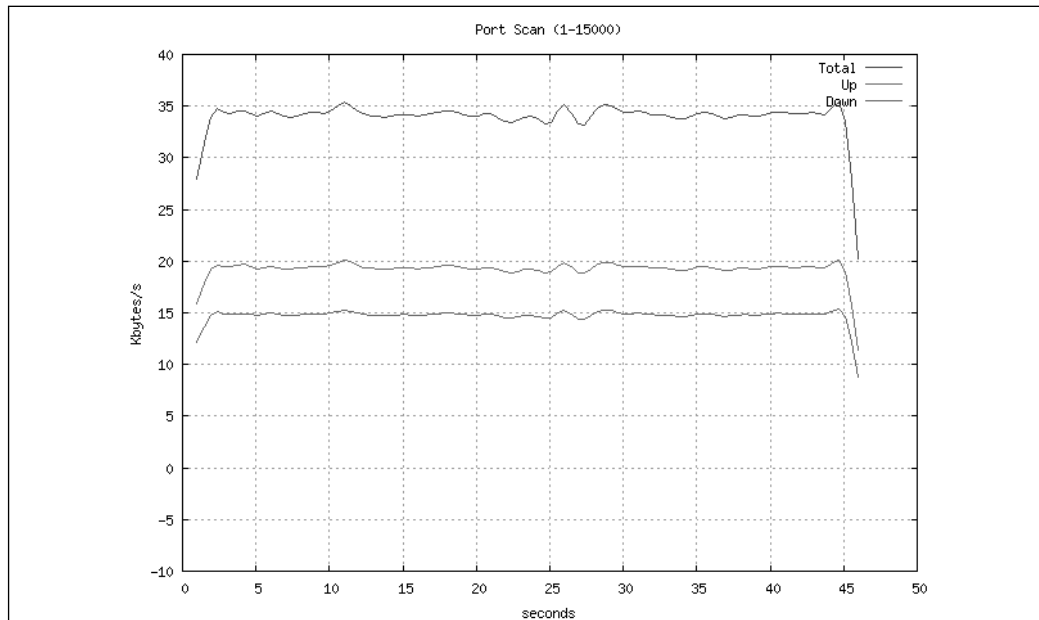
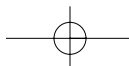
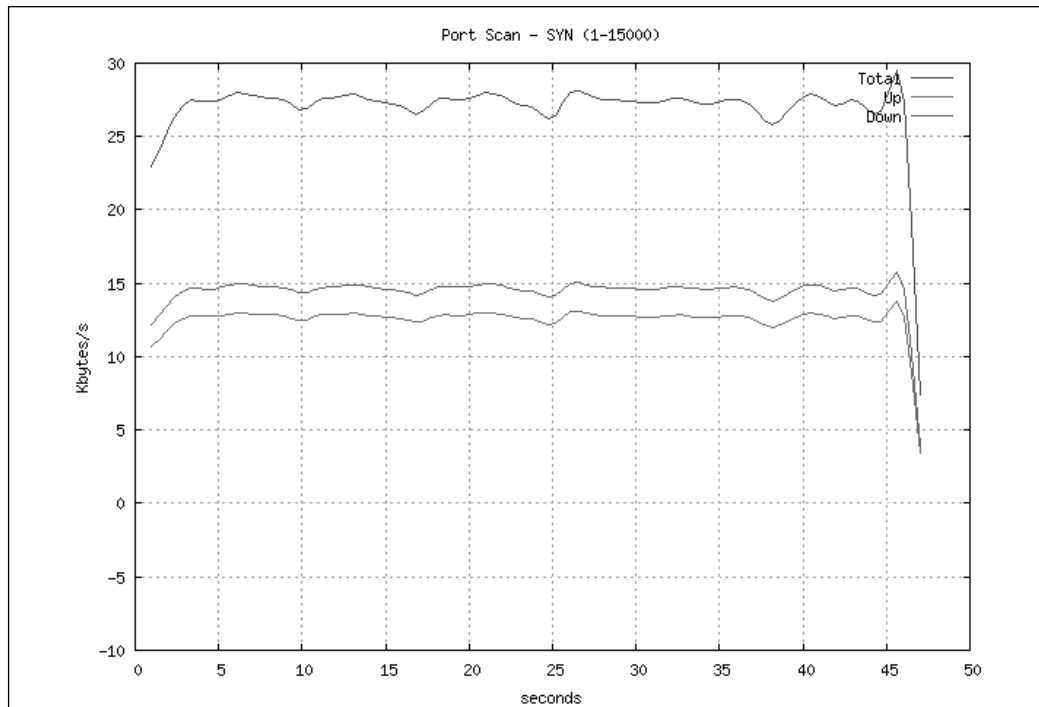
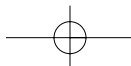


Figure 10.3 Portscan—SYN (1-15000)





308 Chapter 10 • Enterprise Scanning

We can easily control how fast the portscan proceeds and the average bandwidth being consumed by choosing settings other than the defaults Nessus uses for Nmap. Of course, these changes will affect the time it takes to portscan the host, but at the moment, we are more concerned about bandwidth utilization than scanning speed. Note that our goal in enterprise scanning is an optimal configuration where no interaction with the scanning server is required after our early tuning, and thus we can just sit back and receive reports, while Nessus scans at its own speed consuming predetermined bandwidth. The process is automatic, and we don't really care how long it takes, just that the results are accurate and as unobtrusive as possible.

Testing Phase

During this phase, Nessus takes each port reported open by Nmap and runs a service detection process on it to determine the service type of that port (for example, HTTP, SMTP, POP3, etc.). For each service detected, Nessus will then run its arsenal of plugins. The more ports and applications you have on your machine, the more bandwidth consumed. However, since you can control the number of plugins that are run simultaneously, you can easily control the utilized bandwidth during this phase.

By launching Nessus against a default IIS web server and analyzing the traffic generated, we can see that the upstream bandwidth required averages 30KB per second, while the downstream is an average of 239KB per second. Running a similar scan against a default Apache web server will return different results: for the upstream an average of 13KB per second, and for the downstream an average of 73KB per second.

An average is one thing but as the two graphs in Figures 10.4 and 10.5 show, a more elusive bandwidth requirement issue hides behind it, as there are impressive bandwidth peaks going well over the 500KBps for Apache and over the 1,000KBps for IIS. However, in both cases, they are the downstreams—the responses from the server. This means that asymmetric connections can be used while it might appear that this was not the case before.

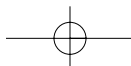


Figure 10.4 Apache Scan (port 80/www)

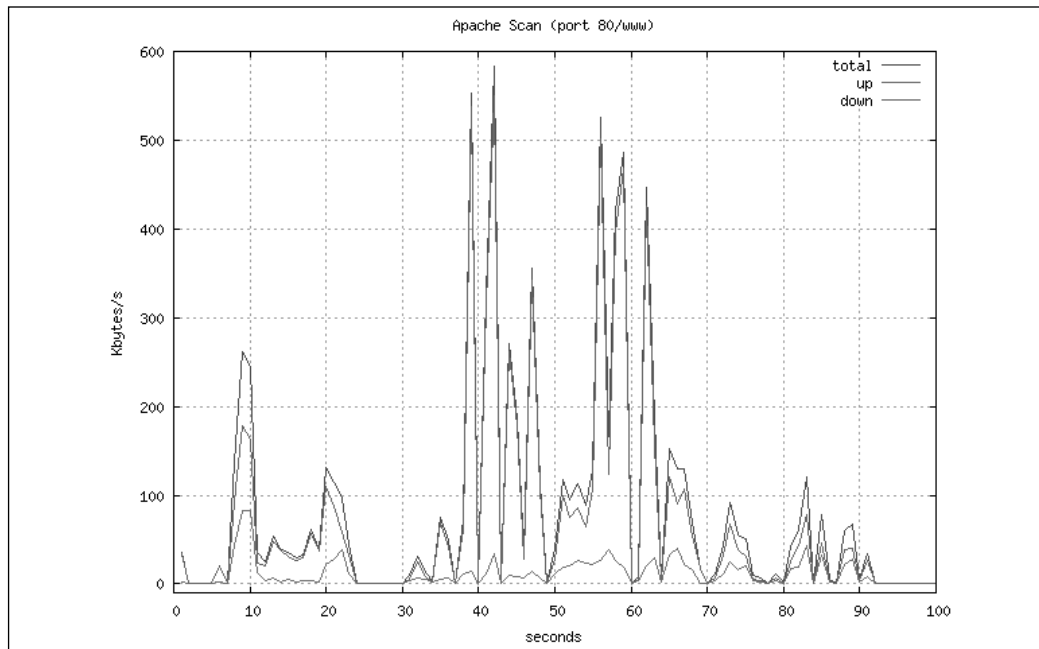
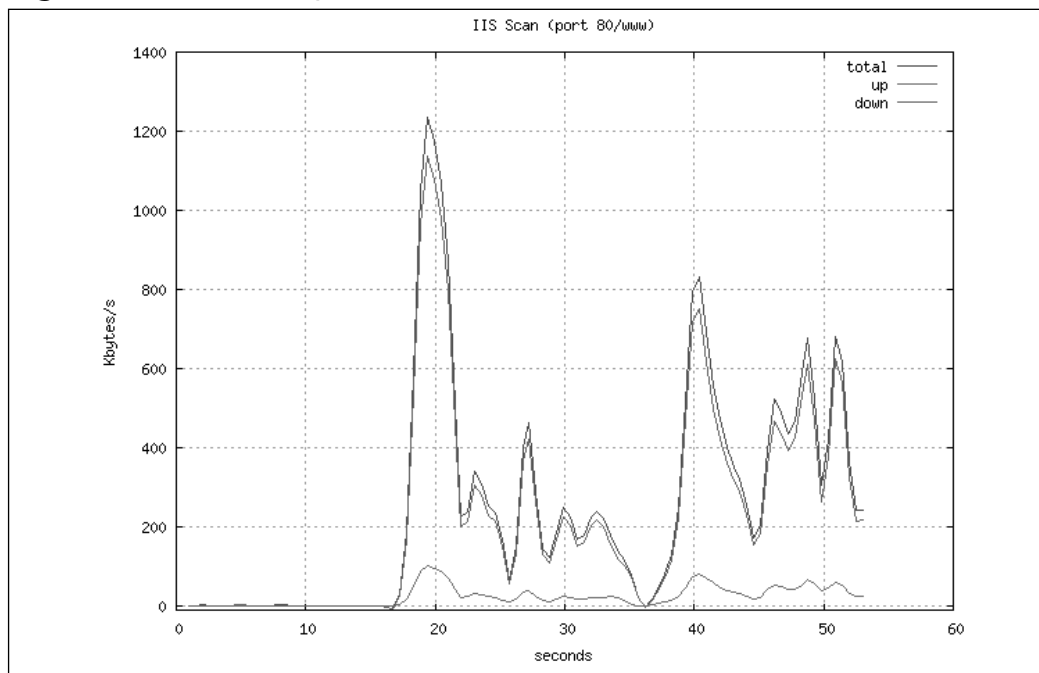


Figure 10.5 IIS Scan (port 80/www)



310 Chapter 10 • Enterprise Scanning

There is one obvious difference between **Apache Scan (port 80/www)** and **IIS Scan (port 80/www)**. The **IIS Scan (port 80/www)** is a slow starter, while the *Apache Scan* appears to consume an average bandwidth throughout the scan.

This difference in bandwidth consumption is due to the type of plugin “sophistication” Nessus uses—when the web server is detected to be Apache, Nessus will run certain tests, whereas if IIS is detected, those tests will not be run.

You should note that HTTP is not a special case—most protocols are just as bandwidth intensive as HTTP, if not more. For the sake of comparison, NetBIOS (the protocol used when scanning for vulnerabilities affecting ports 135, 137, 139, and 445) will yield a higher bandwidth usage on an unpatched Windows 2000 system when it is provided with a username and password combination in comparison to when it is only allowed to use NULL (anonymous) sessions, as shown in Figures 10.6 and 10.7.

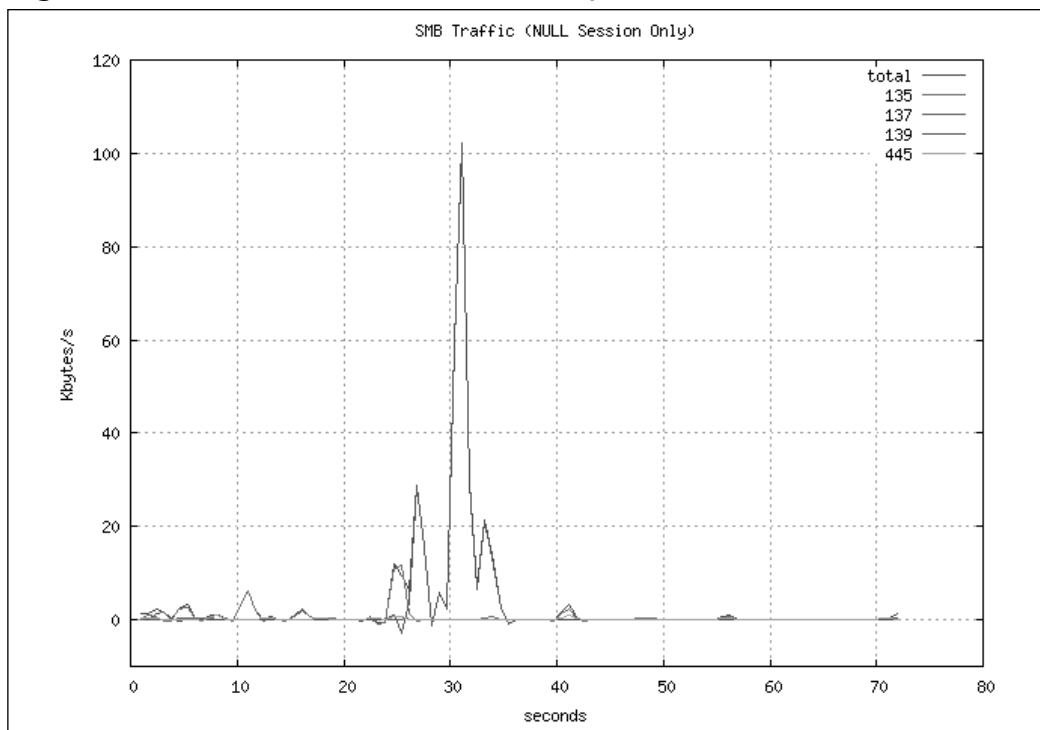
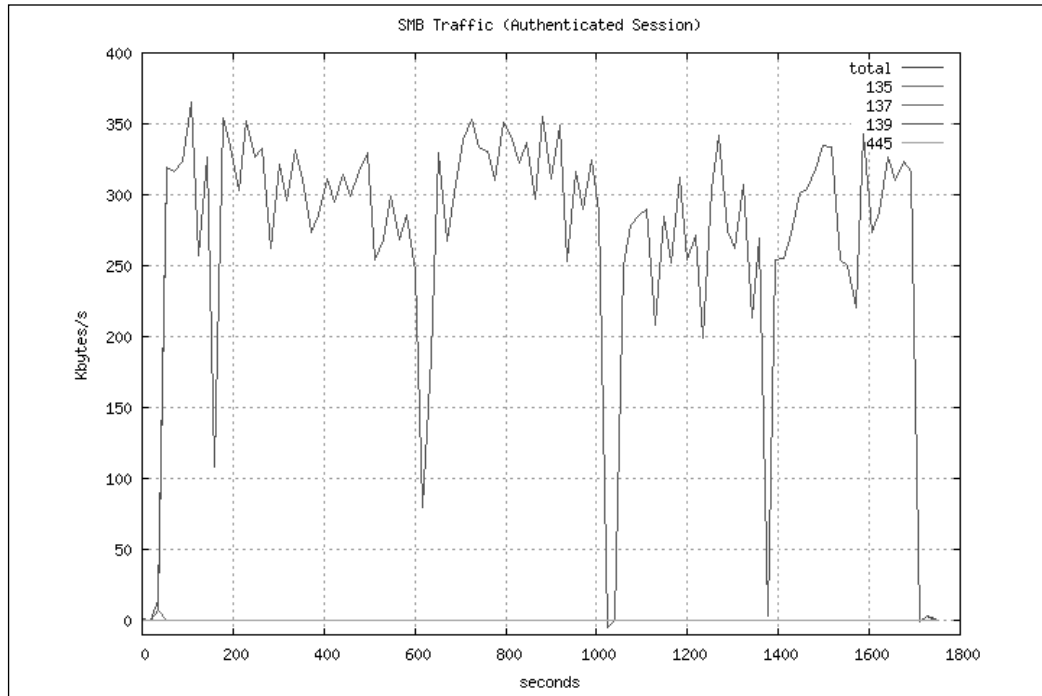
Figure 10.6 SMB Traffic (NULL Session Only)

Figure 10.7 SMB Traffic (Authenticated Session)

The provided username and password allow Nessus to use two important functionalities—registry access and remote share access—which in turn result in more plugins becoming relevant. These two functionalities are time consuming (making the scan take longer to complete) and consume a great deal of bandwidth. However, given valid credentials, they provide very important information; for example, notification of backdoors, worm infections, and policy breaches, including expired passwords, locked accounts, administrative group members, and so forth.

So, what does this all mean? It means that Nessus' bandwidth requirements aren't negligible, and that in most cases they cannot be dismissed as unobtrusive. Furthermore, the different types of operating systems (OSs) and applications being checked affect how much bandwidth is consumed.

There are now two big questions: How can we minimize the average bandwidth being consumed, and where can we place the scanning server without affecting the network's bandwidth utilization?

The first question is easy to answer. You can minimize the average bandwidth consumed by configuring Nessus to run fewer tests in parallel by tweaking the

312 Chapter 10 • Enterprise Scanning

value *Number of checks to perform at the same time* via the graphical user interface GUI or by setting the value of *max_threads* found in *nessusd.conf*.

The best time to employ this tweaking, changing the value of *max_threads*, is whenever there are more than several hosts we want to scan, preferably with the same network mask, or at least on the same physical network (provided they are not separated by network devices or firewalls). Doing so accomplishes two important things. First, it minimizes the strain a scan will have on the network, and second, it minimizes the load on the scanning server itself. At the moment we are more interested in the first goal. The subsequent question will be better explained in the section *Configuring Scanners*, where we explain how to spread the scan over several physical servers, while not necessarily increasing the maintenance workload. We will do this by managing the servers from a centralized location, automating them so they update themselves send the generated reports back to that central location.

Automating the Procedure

Before installing Nessus daemons on several computers, we must first create a test plan that will answer the following questions:

- How do we verify that the scanners do not cause harm to our network?
- How do we monitor the bandwidth usage that might be incurred by our scanning?
- How do we verify that the scanners are working properly and are in fact finding vulnerabilities?

Similar to the Hippocratic Oath taken by physicians, the most important part of the enterprise scanning is to make sure the network does not suffer from being scanned. The network can suffer due to unresponsive applications, high CPU, memory usage, or by traffic congestion. Our scanning servers can cause any of these problems.

These effects will need to be prevented. To avoid problems in applications or bandwidth congestion, you need to prepare some sort of test lab where several nonproduction servers will go through the same scanning as you are planning to use.

The test lab will allow you to verify the effects on your network of setting the port range, plugins to run, network scan speed, and so forth. This test lab will create a scanned environment that is as real as possible, allowing you to detect

issues that might arise in the actual scans and solve them before they can cause problems in your real environment.

The test lab will also ensure that you are not running scans with incorrect settings. For example, running Nessus with a port range of 1–1024 and telling it to regard all other ports as closed will have an adverse effects on the results of your scan. Any service above port 1024 will be considered closed—numerous applications use ports higher than 1024: Oracle, MySQL, MSSQL, IRC, and Proxies, to name a few. Those applications will not be tested for vulnerabilities, and you might never notice.

To avoid such incorrect settings from affecting the results of your scans, you will need to create self-made “honeypots” that you know are vulnerable to a certain extent. These honeypots will be used to verify that Nessus is in fact running correctly. We do not need to actually use commercial honeypots—all we require is to understand our network structure enough and expect certain vulnerabilities to appear. For example, if we know a certain Windows machine wasn’t patched with the latest MSXX-XXX patch, we can use one of Nessus’ tests as a marker that our scans are working correctly and that we are able to access the registry to detect that missing patch.

We assume from now that you created your test lab, ran some Nessus scans, verified that they are not affecting your test lab’s network, and you are running them as correctly as possible finding all the vulnerabilities that appear in the test lab servers. Does this mean that we are finally ready to scan the enterprise? Again, we must reiterate that you are still missing a crucial part of making Nessus work correctly—this important part is what makes the difference between recurring vulnerability assessment and one-time vulnerability assessment.

One of Nessus’ most important features is its capability to update tests amazingly frequently. Nessus is installed with an easy-to-use script that will update the plugin library used by the scanner. The script, trivially called `nessus-update-plugins`, downloads a file from the www.nessus.org web site, extracts its content to the plugin folder, and restarts the Nessus daemon, making the update process as smooth and automated as possible.

However, the script’s behavior leaves more to be desired. How can you verify that the update process was successful for every Nessus daemon you have on the network? How can you make sure all Nessus daemons are in fact up to date? How can you roll out any custom-made plugins you create? How can you avoid the Nessus daemons downloading a big file from the Internet, possibly slowing

314 Chapter 10 • Enterprise Scanning

the Internet connection? And, how can you avoid giving direct access to the Internet to each of these Nessus daemons?

All of these questions can be answered using a more centralized approach to updating the Nessus engine. The required materials for centralizing the plugin-updating mechanism are a web server and some scripting skills. The update mechanism is composed of five stages:

1. Download the update.
2. Unpack the update.
3. Install the update.
4. Restart the Nessus daemon.
5. Report success.

Each stage will be done as much as possible in a single operation, making the update mechanism as redundant and safe as possible. In addition, it will guarantee that if a stage fails, the centralized server is informed. To implement the download stage, we suggest you either use the `wget` utility (which features the ability to Proxy, HTTP Authenticate, Resume, Retry, and many other nifty features) or implement a file downloading script via some external libraries, like Perl's amazingly popular LWP module.

We suggest you modify the downloading process so it does not download the same file each time it updates, as `nessus-update-plugins` does, but rather have some type of versioning mechanism where you can keep track of what version the Nessus daemon last attempted to update.

The version does not have to be more than a simple counter increasing by one each time you decide to roll out an update to your Nessus daemons, or perhaps a timestamp. After the download is complete, the updating mechanism will inform the web server that it was successful in downloading the update.

Our next stage is extraction and installation of the update. You do not have to separate the two stages, although it is recommended that you verify that the extraction was successful before trying to overwrite files—some extraction programs will partially overwrite the files if the archived file was corrupted, making your installation unstable or even unusable.

The installation of new plugins (unlike overwriting existing plugins) has no effect until the Nessus daemon is restarted; therefore, we must restart the Nessus daemon before continuing.

Restarting Nessus via a HUP signal (as is done by `nessus-update-plugins`) has no effect on the scan that is currently in progress—the scan will continue scanning as if nothing had happened.

This is both good and bad. It's good because it doesn't stop our current scan, but it is bad because it doesn't reload any new scripts, making the current scan only partial (without all the new or updated plugins). We cannot avoid this, but we can modify our update mechanism to recognize such an event and alert us that a scan was in progress while the update was running, so that we can make a decision on whether we want to rerun that scan.

You can easily verify whether a scan is currently in progress by obtaining the list of running processes on the Nessus daemon and looking for the string `testing` (followed by the IP being tested). If this string appears, a scan is currently in progress. If not, you can safely signal a HUP at the end of the update mechanism or even restart the Nessus daemon service.

Once the Nessus daemon restart process has completed by either a HUP signal or a full-fledged stop and start, the script should again report to the web server that it has successfully restarted Nessus. The cautious administrator might also want to verify that Nessus has successfully HUPed/restarted by checking the `/var/log/messages` (or any other file configured by the `nessusd.conf` file) for the string `nessusd x.x.xx started` (if Nessus was started/restarted) or for the string `Caught HUP signal - reconfiguring nessusd` (if Nessus was HUP'ed).

Unfortunately, we can't cover the aspects of automatically updating each Nessus daemon, including its libraries, and binaries using a mechanism similar to the one discussed previously, as it is too complex to be covered here. However, we can suggest using other means of updating that are equally good and relatively safe. By using Debian's `apt-get`, you can constantly check whether the Nessus you are using is up to date. Debian's unstable distribution is updated with the latest version of Nessus. At the time of writing, Nessus 2.0.10 is available by Debian. By simply running the command **`apt-get -t unstable update`**, followed by **`apt-get install -t unstable nessusd nessus`**, whenever a new version is available it will be installed on the machine.

Nessus' client provides two different ways of determining configuration and version information on a remote Nessus daemon. One is the **`-p`** parameter that allows you to obtain a list of the server and plugin preferences. The parameter can also work from a remote location where the only requirements are a username and password or client-side certificate (depending on the authentication mechanism used by the Nessus daemon). The other is the **`-P`** (uppercase) param-

316 Chapter 10 • Enterprise Scanning

eter that allows you to obtain a list of plugins installed on the server. This parameter, as the previous one, can be used from a remote location.

Instead of using the **-p** and **-P** parameters on your own and parsing the results returned, we recommend using the `update-nessusrc` script (available at www.tifaware.com/perl/update-nessusrc/). This easy-to-use Perl script takes an existing configuration file and is able to display the differences between your configuration file and the remote Nessus daemon. The differences include version changes, different plugin settings, and new, removed, or changed plugins. Unfortunately, the `update-nessusrc` script does not support command-line parameters that control the host to which it connects or change the username and password combinations it uses. However, by modifying the `update-nessusrc` script and adding support for such a parameter, you can make the script incredibly handy.

The Nessus daemon keeps track of its actions through the log files. These log files can therefore be used to maintain a good record of what was scanned, when it was scanned, and how long it took. As going over the Nessus daemon log file manually is no easy task, we suggest using `nessustail.pl` (available from Nessus' `nessus-tools` package). The tool is a very easy-to-use Perl script that will comb through the message log created by the Nessus daemon and highlight potential problems such as segfaults, interruptions, HUP signals, and plugins that were too slow to finish.

Configuring Scanners

We'll now look at specific ways to deploy distributed scanning in terms of scanning topologies, examining the advantages and disadvantages of each.

Assigning the Tasks

Once we have decided what we want to scan, we need to start dividing the scans between the different hosts. When we divide the scans, we need to make sure we do not breach the confidentiality of the different departments by placing a single server scanning administratively separate networks, we do not cause too much traffic across the network by placing the server in a single place and scanning the whole network from it, and that we can still control the servers placed around the network even if they are in the different parts of the network.

There are three possible distributed scanning topologies you can use: *Star*, *Flat*, and *Islands*. Each topology has its advantages and disadvantages. We will start

with the islands topology, as it is the easiest to explain. The islands' goal is to install Nessus daemons that are completely isolated from each other, thus maintaining the highest form of separation between the different departments. In the islands topology, there is no single point of control to the servers, and each network has its own Nessus client connecting to the server.

Advantages of the islands topology include:

- Information cannot leak between departments.
- Any problems with one server's scans does not affect the others.
- Each server can have an independent administrator. This administrator doesn't gain any additional access to other servers or other scanning servers.
- No additional firewall or networking devices rules need to be placed between the departments or between the Nessus client and the server.
- The different servers can provide different points of view of the same hosts (for example, DMZ vs. MZ).

Disadvantages of the islands topology include:

- Maintenance overhead because there is no centralized management.
- Higher hardware costs, as more servers are required.
- No centralized updates server can be created.
- Data cannot be correlated between different servers, or brought in to a centralized database for report consolidation.

The next topology we will discuss is the flat topology, which provides real increases in scalability over the islands topology, although it raises bandwidth requirements. In the flat topology, the Nessus daemons are installed all over the network. The network itself is fairly open between the departments and allows traffic to flow unobstructed. In this type of topology, the Nessus client can be used to manage the servers practically from any location. As the network is wide open, a single Nessus daemon can scan several departments.

Advantages of the flat topology include:

- A single server can scan the entire network.
- Management can be done virtually from any point of access on the network.

318 Chapter 10 • Enterprise Scanning

- A centralized update server can be used to update all the different Nessus daemons.
- Reports can be consolidated between the different servers.
- No additional firewall or networking devices rules need to be placed between the departments or between the Nessus client and the server.

Disadvantages of the flat topology include:

- Information may leak between two reports, as servers can house more than a single network vulnerability report.
- As a single server can be used to scan the entire network, it becomes a single point of failure with regard to the enterprise's vulnerability scanning.
- A single server cannot provide different views of the network's vulnerabilities (for example, DMZ vs. MZ).
- A single server causes higher bandwidth consumption across the organization compared to distributed servers.

The last topology we will discuss is the star topology. In this topology, the Nessus daemons are servers spread cross the organization that are all connected (in addition to their normal network access) to a management network. In this topology, we can manage all the servers from the management network, and can use this network to transfer data between the servers and to update these servers. However, the servers themselves cannot interconnect, as the management network allows only access to the centralized network.

Advantages of the star topology include:

- Information cannot leak between departments.
- Bandwidth consumption is divided between the servers.
- It provides different views of the network's vulnerabilities (for example, DMZ vs. MZ).
- A centralized update server can be used to update all the different Nessus daemons.
- Reports can be consolidated using a centralized server housing all the data.

Disadvantages of the star topology include:

- Management can only be done from a single point.
- Higher hardware costs are incurred, as more servers are required.
- Reports cannot be consolidated by interconnecting two servers.
- Additional firewall and network devices rules need to be placed between the departments and the management network.
- A single administrator would have control over the entire set of Nessus daemons, with the ability to inadvertently breach the different departments' confidentiality.

System Requirements

The Nessus daemon by itself isn't a large memory or CPU consumer. An idle Nessus daemon, with no clients attached, will consume virtually no CPU time, and around 1MB of memory. A disconnected Nessus client memory consumption is around 1.5MB. Once a connection is made, the daemon's memory consumption will jump to around 2.5MB, and the client's memory consumption will jump to around 5.5MB. Any additional Nessus client connecting to the daemon will consume an additional 1.5MB of memory.

Using these figures, we can safely assume that the Nessus daemon can hold a few dozen open connections with Nessus clients. However, the numbers given do not provide an accurate picture, as they do not include the memory consumption requirements needed once a scan starts. The memory consumption will vary greatly during the portscan phase and the plugin running phase, as different plugins are loaded, executed, and unloaded.

The Nmap process requires around 3.5MB of memory for each host it portscans with TCP connect() scan and OS fingerprinting. By default, Nessus issues scans for up to 255 hosts in parallel, which means that a heavily used network can easily consume all available memory during the portscanning phase.

This calculation doesn't take into account the fact that the Nessus daemon will spawn an additional process for each host it scans, consuming roughly an additional 1.5MB of memory per host scanned. This process is responsible for launching the different plugins (the NASL and NES files).

Running the Nessus client with a Nessus daemon, where we configured the Nessus daemon to run one plugin at a time, will cause the Nessus daemon to

320 Chapter 10 • Enterprise Scanning

consume roughly 10MB of memory in total. Using the same configuration, but configuring it to run two plugins at a time, will make the memory consumption jump at times to 12MB, but the average memory consumption will remain around the 10MB mark as Nessus waits for the two plugins it has launched to finish prior to launching two new ones. This is true even if the first plugin finishes several minutes before the second plugin .

Table 10.2 illustrates the memory consumption of the Nessus daemon vs. the number of simultaneous plugins being used (when scanning a single host).

Table 10.2 Memory Consumption of the Nessus Daemon

Number of Plugins	Peak Memory Consumption	Average Memory
1	10MB	10MB
2	12MB	10MB
3	17MB	12MB
4	21MB	16MB
8	35MB	28MB
16	38MB	32MB
32	77MB	50MB
64	92MB	60MB

NOTE

You will need to modify the value of **max_checks** in the `nessusd.conf` file to go over the 10 plugins mark. To go over the 64 marker, you will further need to modify the `nessus-core/nessusd/pluginlaunch.c` file and change the value of `MAX_PROCESSES` from its default value of 32 to whatever you desire and then recompile.

The calculation for more than one host is a bit more difficult, and requires a long trial and error process to get the memory consumption requirements. As a rule, the **nessusd** process requires an average of about two times as much memory for any additional host for the peak memory requirement, and about one and a half times the memory needed for the average memory. We do recommend, however,

as memory is a relatively inexpensive component, to equip the scanning servers with at least 1GB of RAM to answer most of your memory needs for scanning. You need to remember that you do not want to scan too many hosts in parallel, or use too many plugins in parallel, as both of these are bandwidth consumers. The Nessus daemon consumes plenty of CPU but it is not a CPU hog, as the utilization of CPU consumption depends on two main factors: the number of hosts being tested and the number of plugins being launched. As more hosts are tested and more plugins are launched in parallel, the CPU time for processing of the network data increases, and with it, the CPU loads. As in many other tasks, increasing memory has a greater effect on Nessus' speed than using a faster processor.

NOTE

The Nessus daemon does not support Symmetric Multi Processing (SMP) in its native code, and will not benefit from it more than any other non-native-SMP program running on the computer. We therefore recommend running the Nessus daemon on a dedicated machine, instead of installing it on a multipurpose server.

Scanning for a Specific Threat

Every once in a while, a new threat arises in the form of a critical advisory. The new vulnerability may affect more than one host on your network, but you are not certain which ones. It would appear that your viable solution would be to scan your entire network for all vulnerabilities and filter out that new vulnerability in which you are interested. However, scanning the entire network is neither simple nor quick.

We suggest this alternative. Scan your network with a limited set of plugins, which allows you to scan your entire network in a very short timeframe while trimming down the amount of vulnerabilities found to just those about which you are concerned. The only shortcoming of this type of scan is that you will need to make sure that all the plugin dependencies are met—open ports, dependent plugins, and so forth.

The dependencies requirement can easily be met by setting the parameter **auto_enable_dependencies** to **yes** in your configuration file and choosing the plugin numbers you are interested in enabling. As an alternative to changing your

322 Chapter 10 • Enterprise Scanning

configuration file, you can use the tool `update-nessusrc` mentioned in the previous section. One of the tool's parameters is **--includes** followed by ID numbers. These ID numbers are the plugin IDs. Plugin IDs are unique identifiers for the specific test you are interested in using. For example, if we want to test for Microsoft's MS04-011 (incidentally, this patch addresses the vulnerability that the Sasser worm exploits), we need to first locate the appropriate plugin filename. In our case, there are several related plugins—we choose to use the registry-based `smb_nt_ms04-011.nasl`. Inspecting the file `smb_nt_ms04-011.nasl`, and looking for the entry **script_id** will reveal that the script's ID is 12205. We will use this number to run **update-nessusrc -c "" -f "" -r "" -i "12205" basic**, where **basic** is the default Nessus configuration file, **12205** is the relevant script ID, **-c** sets the name of the categories you want to enable, and **-f** sets the name of the families you want to enable.

NOTE

Remember that before you start scanning using this configuration file, you need to set the **auto_enable_dependencies** to **yes**. Failing to do so will return scan results that are misleading, as some of the tests plugin 12205 depends on will not execute.

In some cases it is better to meet (or avoid meeting) the plugin dependencies on your own, as the dependencies directly affect the behavior of certain plugins and the “path” (or method) that is used to discover a certain vulnerability. However, this is only recommended to those who are fully aware of the consequences of not meeting a given dependency—the most problematic outcome is that you might completely miss the presence of the vulnerability you were looking for.

Notes from the Underground...

Testing New Plugins

When you want to test your newly acquired or created plugin that depends on other plugins for such things as registry access, the NASL command-line interpreter will be insufficient, as it does not support Knowledge Base access or meeting of dependencies. The only viable solution in this case would be to run that test using a configuration file that specifically requests this newly formed plugin as the only plugin to execute, and that you mark the **auto_enable_dependencies** to **yes**.

We can easily illustrate the difference between meeting the dependencies and failing to do with an example of plugins that have the **script_exclude_keys** entry in them. Each plugin that has the **script_exclude_keys** entry in it will not be executed if a plugin has previously set the key this entry lists to exclude, such as in the case of **tcp_chorusing.nasl**.

NOTE

The **tcp_chorusing.nasl** plugin checks whether a remote Windows 95/98/Me machine answers to the same packet more than once—this will happen if it has more than one TCP/IP stack bound on that adapter.

The **tcp_chorusing.nasl**'s plugin logic is as follows: If the key **SMB/WindowsVersion** (which **tcp_chorusing.nasl** excludes) is set, the remote host is a Windows XP/2000/2003, and as such is certainly not vulnerable to this attack. However, if the key is not set, it will go about testing the vulnerability against the remote machine, using a real attack packet. Therefore, plugins partially answering to the dependency required by the test (plugins that set a value to **SMB/WindowsVersion**) will cause the plugin to test each host it finds on the network for the vulnerability.

In addition to the exclude behavior, certain plugins read Nessus' internal information Knowledge Base and act according to the information given there. Therefore, if we want to control the plugin's behavior, we will need to prevent

324 Chapter 10 • Enterprise Scanning

the Knowledge Base information from being written. One way of doing this is to stop certain plugins from running by not meeting or only partially meeting their dependencies.

Notes from the Underground...

The Nessus Community and Plugins

Unlike other commercial vulnerability assessment tools, Nessus enjoys a large community of plugin developers and plugin maintainers. However, as there is no paid quality assurance team making sure the plugins are accurate, false positive free, and informative, the Nessus community requires your assistance. The most helpful feedback you can provide the Nessus community is its ability and inability to detect vulnerabilities, and, if possible, new plugins that detect those vulnerabilities. It is important to note that if you do find a problem with any of the Nessus plugins, you should report as much information regarding the host being tested and e-mail all the information to the bugs@nessus.org.

Best Practices

In the following section we will outline the best practices you should keep in mind.

Divide and Conquer

It is important to avoid sensitive information unintentionally appearing in reports that are available to less-classified personnel. Both the Nessus configuration file and the reports generated might contain sensitive information such as usernames, passwords, domain names, and so forth. As different divisions of your company are exposed to different types of sensitive information, you should run a separate scan for each division to keep reporting tools separate.

Segregate and Limit

In some companies, a different person from the one who installed the Nessus daemon might control the Nessus client. In such cases where it is not possible to

further separate the Nessus daemons between the different networks, it is important to create daemon-based rules to prevent users from scanning networks and hosts that they don't need to.

The Nessus daemon's internal configuration file allows creation of rules that allow this segregation and limitation to be done easily. A rule is basically composed of an action: *reject*, *accept*, and *default* followed by an IP and associated network mask. The rules can be serverwide, user based, or client based. As we do not have control over the client-based rules, you should mainly concentrate on server-wide and user-based rules.

NOTE

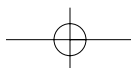
Client-based rules are set by the Nessus client and offer no protection from someone using your Nessus daemon to scan a host it shouldn't.

Serverwide rules block a certain range of IPs from being scanned by the server. They are relevant to all users who connect to the Nessus daemon, and cannot be overridden by choosing a different user. User-based rules block a certain range of IPs, as before, from being scanned by the server, but they are dependent on the user who logged on to the system. A certain user might be able to scan host A, and another might not—it all depends on the username used to log on to the Nessus daemon.

We recommend placing a limited range of IPs that hosts can scan on each of the Nessus daemons you install, and extending that range as required. These rules will prevent the scanning host from accidentally being used to scan the entire network or even a single host you didn't intend to scan, such as sensitive servers, production networks, and so forth.

Certificates for the Forgetful

The Nessus daemon supports two authentication methods: password-based and certificate-based. We will not discuss why one is better than the other, but from our experience, certificates are easier to move and handle, and are more secure against snooping and capturing, while passwords can be captured, guessed, forgotten, and so forth. As such, we recommend that you install your entire Nessus daemon installation base with certificate authentication instead of password-based



326 Chapter 10 • Enterprise Scanning

authentication. Both have no implications on anything but the logon phase where the Nessus client connects to the Nessus daemon.

Speed Is Not Your Enemy

Having a faster connection with lower latency between your scanning server and the host being scanned improves the speed at which you can scan the host and the reliability of your results. A host scanned over a low-bandwidth WAN connection with a high percentage of packets lost will produce a greater percentage of both false positives, usually from denial-of-service (DoS) tests, and false negatives, which generally come with web-based tests.

It is therefore important to make sure that you conduct your scans through an optimal network connection, and that this connection offers as much bandwidth as you require and introduces as little latency as possible.

Keep a Watchful Eye

It is easy to lose track of the last time you received the results from a particular scan. Sometimes, a scan will only return partial results. This is especially true when you are working with large-scale networks where a momentary reset of a router, firewall, or other network device might cause hosts to disappear, artificially thinning your reports.

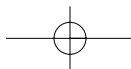
It is critical to always be aware of what vulnerabilities have been fixed and what new vulnerabilities are present as a result. Tracking these vulnerabilities and including them in the reports you receive can help you make certain that everything is working properly.

Data Correlation

In this next section, we'll focus on data correlation, especially in regards to the reports that are developed.

Combining Reports

Obviously, the most important part of Nessus isn't the scan itself, but rather the scan's results. While reading dozen of Nessus reports using the Nessus GUI can prove difficult, you can easily solve this problem by building your own vulnerability database. A comprehensive database often contains reports about all of the servers that were found to be vulnerable, along with the vulnerability identification numbers, descriptions, and risk factors. Once you've entered the data into a



database, it's easy to query it and reveal which parts of the network are the most vulnerable, which vulnerabilities affect most of your servers, which servers contain a particular worm's affected vulnerabilities, and so forth.

Preparing Your Database

Before we can begin, we will need to prepare the database. For this section, our examples will use MySQL as our database server, because it is freely available for both the Linux and Windows operating systems, and is easy to use. However, any other database server can be used, instead. No matter which database server you choose, it will need to hold at least one database and one table where your information can be stored. You can easily parse Nessus' NBE (Nessus BackEnd) file format with any program that can interpret content that is delimited by the pipe sign (|).

The fields the Nessus' NBE returns are IP address, affected port, script ID, vulnerability type, and description. To begin, we will first need to create a table that contains these fields. We will start by using the following SQL query:

```
CREATE TABLE ScanResults (  
    ID INT AUTO_INCREMENT,  
    IP TEXT,  
    Port TEXT,  
    PluginID INT,  
    Type TEXT,  
    Description TEXT,  
    PRIMARY KEY ID (ID)  
);
```

You might need to modify the script depending on your database configuration.

The aforementioned table will be used to hold our scan results, which we will parse using the following Perl script:

```
#!/usr/bin/perl  
# nbeparser.pl - Nessus NBE parser  
# Thanks to A.M.I for helping with the Voodoo  
use strict;  
use DBI;  
  
my $db = "nessusdb";
```

328 Chapter 10 • Enterprise Scanning

```
my $dsn =
"DBI:mysql:database=$db;host=localhost;port=3306;mysql_read_default_file=/etc
/mysql/my.cnf";
my $user = "nessusdb";
my $pass = "nessusdb";

my $dbh = DBI->connect($dsn, $user, $pass, {'RaiseError' => 1});

my $filename = shift;
open(NBE, "$filename") || die "File not found\nYou need to provide this
program with a valid filename to parse.\n";

while (<NBE>)
{
    my @values = split(/\|/, $_);

    my $SQL = "INSERT INTO ScanResults SET IP='".$values[0]."',
Port='".$values[1]."', PluginID='".$values[2]."', Type='".$values[3]."',
Description=".$dbh->quote($values[4])."';

    print "$SQL\n";

    my $sth = $dbh->prepare($SQL) or die "Cannot prepare statement:
$DBI::errstr\n";
    $sth->execute() or die "Cannot execute statement: $DBI::errstr\n";

    $sth->finish();
}

$dbh->disconnect();
close(NBE);
```

You can run this script on a system that has a Perl interpreter installed by typing the following at a command line. Root or administrator privilege will not be necessary, so make sure to run this command as an ordinary user:

```
nessus-user $ ./nbeparser your_real_report_name.nbe
```

If the script has successfully completed, you should now have your own Scan Results table that contains the outcome data of the Nessus report file whose name you specified on the command line; in this case, **your_real_report_name.nbe**. From here on, you should be able to query your database and find the information that you need from this report.

This script is not a perfect solution, however. It is, in fact, a very simplistic piece of code and is therefore unable to handle tasks that are more complex. For more complex (especially enterprise-scalable) applications, you'll need to write your own. We'll outline the limitations here to help you realize when you've outgrown the script. First, this script can't verify whether the results you seek are already there. Moreover, it can't tell you the date that your results were generated. Additionally, the resulting database table does not differentiate between different reports' filenames. Therefore, a unique line is generated for any entry found in the results file. Thus, if you accidentally run the script on the same results file, duplicate entries will appear in your database. What this comes down to is that the table is too simplistic to support any form of differentiation of data. This means that it is difficult to discern which vulnerabilities are old problems and which ones have recently appeared.

Outside of differentiation, this script will not work in a model where you scan the same targets from multiple points. For example, it isn't possible to show the same target's vulnerability results from the DMZ and the privileged network. In short, it's just a basic structure to get you started.

Adding additional columns can solve some of these problems, but before we do that, let's look at some queries we can execute to find useful information from the database. Let's start by finding which vulnerability affects the most computers on our network. We first need to find out what script IDs we have affecting our network, so we will execute:

```
select DISTINCT PluginID from ScanResults;
```

```
+-----+
| PluginID |
+-----+
|    12205 |
|    10394 |
|    10400 |
|    10150 |
|    11011 |
+-----+
```

330 Chapter 10 • Enterprise Scanning

We then need to take each PluginID and count the number of entries it has in the database. We will use the following SQL statement:

```
select COUNT(*) from ScanResults WHERE PluginID='11011';
+-----+
| COUNT(*) |
+-----+
|         8 |
+-----+
```

After going through all the different PluginIDs, we can conclude that PluginID 11011 is the one that affects the most machines in our network.

If we look up this plugin's description, we learn that this test detects whether the remote servers support NetBIOS, and that we forgot to eliminate the same host being affected more than once by this vulnerability. We will recount using this SQL statement:

```
select COUNT(DISTINCT IP) from ScanResults WHERE PluginID='11011';
+-----+
| COUNT(DISTINCT IP) |
+-----+
|                   4 |
+-----+
```

It seems almost all of our vulnerabilities appear in all four hosts we scanned, and that we don't have a single vulnerability that we need to give special attention to before the others. This is in fact false—if we take into account the severity of the vulnerabilities, one vulnerability will be more important to highlight than the others.

Let's start by finding how many High/Medium/Low risk vulnerabilities we have in the database. For the sake of simplification, we will consider a *Serious* risk factor as equal to *High*. We can count the number of vulnerabilities according to their risk factor by executing the following SQL statement:

```
select COUNT(*) from ScanResults WHERE Description LIKE '%Risk factor :
High%' OR Description LIKE '%Risk factor : Serious%';
+-----+
| COUNT(*) |
+-----+
|         1 |
+-----+
```

```

+-----+
select COUNT(*) from ScanResults WHERE Description LIKE '%Risk factor :
Medium%';
+-----+
| COUNT(*) |
+-----+
|      4   |
+-----+

select COUNT(*) from ScanResults WHERE Description LIKE '%Risk factor :
Low%';
+-----+
| COUNT(*) |
+-----+
|      1   |
+-----+

```

We can easily see that we have a High/Serious risk factor vulnerability. We can cross-reference this by counting the vulnerabilities that occur more than once on our network by using the SQL **UNION** statement. We will come back to this in the next section, which covers differential scanning.

Now that we know we have a High/Serious risk factor vulnerability, we can pull its PluginID and search for an explanation about this vulnerability either from the description that comes with that entry in the database or by going to <http://cgi.nessus.org/plugins/dump.php3?id=XXXXXX> and replacing XXXXX with the PluginID.

```

select PluginID from ScanResults WHERE Description LIKE '%Risk factor :
High%' OR Description LIKE '%Risk factor : Serious%';
+-----+
| PluginID |
+-----+
|  12205   |
+-----+

```

Going to <http://cgi.nessus.org/plugins/dump.php3?id=12205> shows that the plugin that reports this vulnerability is named *Microsoft Hotfix KB835732 (registry check)*, and that this problem is solved by installing the patch available from www.microsoft.com/technet/security/bulletin/ms04-011.msp.

332 Chapter 10 • Enterprise Scanning

One of the greatest benefits of using a database is that you can easily compare two different results that were generated from two different network locations; for example, DMZ vs. privileged network. We first need to modify our database table to include an *Exposure* column, and insert the results gathered from the different sections of the network.

We start by adding a new column named *Exposure*:

```
ALTER TABLE ScanResults ADD Exposure TEXT;
```

We will mark all existing records with the word *Privileged*, using the following statement:

```
UPDATE ScanResults SET Exposure='Privileged';
```

Now we should restart the scan, and insert the results using this script. However, before we rerun the scan, we will add to the SQL statement the string:

```
Exposure='DMZ'
```

Resulting in:

```
my $SQL = "INSERT INTO ScanResults SET IP='".$values[0]."',
Port='".$values[1]."', PluginID='".$values[2]."', Type='".$values[3]."',
Description='.$dbh->quote($values[4]).', Exposure='DMZ'";
```

We will now have the same table as before, but with new entries that illustrate the state of vulnerability from an internal network (privileged) and from an external network (DMZ). The first neat thing we can do is compare the results we received from the privileged network with that of the DMZ and see which vulnerabilities we solved by segmentation alone.

The following SQL statement can be used to find out which vulnerabilities “disappeared” due to scanning from the two different locations:

```
SELECT CurrentScan.IP, CurrentScan.Port, CurrentScan.PluginID FROM
ScanResults AS CurrentScan LEFT JOIN ScanResults ON (CurrentScan.PluginID =
ScanResults.PluginID AND CurrentScan.Port = ScanResults.Port AND
CurrentScan.IP = ScanResults.IP AND CurrentScan.Exposure !=
ScanResults.Exposure AND ScanResults.Exposure = 'DMZ') WHERE
CurrentScan.Exposure = 'Privileged' AND ScanResults.Exposure IS NULL;
```

```
+-----+-----+-----+
| IP           | Port                   | PluginID |
+-----+-----+-----+
| 192.168.1.4  | netbios-ssn (139/tcp) | 11011 |
| 192.168.1.13 | netbios-ssn (139/tcp) | 11011 |
```

```
| 192.168.1.138 | netbios-ssn (139/tcp) | 11011 |
| 192.168.1.243 | netbios-ssn (139/tcp) | 11011 |
```

```
+-----+-----+-----+
```

The following SQL statement will reveal which new vulnerabilities have “appeared” due to scanning from the two different locations:

```
SELECT ScanResults.IP, ScanResults.Port, ScanResults.PluginID FROM
ScanResults AS CurrentScan LEFT JOIN ScanResults ON (CurrentScan.PluginID =
ScanResults.PluginID AND CurrentScan.Port = ScanResults.Port AND
CurrentScan.IP = ScanResults.IP AND CurrentScan.Exposure !=
ScanResults.Exposure AND ScanResults.Exposure = 'Privileged') WHERE
CurrentScan.Exposure = 'DMZ' AND ScanResults.Exposure IS NULL;

Empty set (0.00 sec)
```

The following SQL statement will reveal which vulnerabilities “persist” between the two different locations:

```
SELECT ScanResults.IP, ScanResults.Port, ScanResults.PluginID FROM
ScanResults AS CurrentScan LEFT JOIN ScanResults ON (CurrentScan.PluginID =
ScanResults.PluginID AND CurrentScan.Port = ScanResults.Port AND
CurrentScan.IP = ScanResults.IP AND CurrentScan.Exposure !=
ScanResults.Exposure AND ScanResults.Exposure = 'DMZ') WHERE
ScanResults.Exposure = 'DMZ' AND CurrentScan.Exposure = 'Privileged';
```

```
+-----+-----+-----+
| IP          | Port                | PluginID |
+-----+-----+-----+
| 192.168.1.243 | microsoft-ds (445/tcp) | 12205 |
| 192.168.1.13  | microsoft-ds (445/tcp) | 10394 |
| 192.168.1.243 | microsoft-ds (445/tcp) | 10400 |
| 192.168.1.4   | microsoft-ds (445/tcp) | 10394 |
| 192.168.1.4   | netbios-ns (137/udp)   | 10150 |
| 192.168.1.243 | microsoft-ds (445/tcp) | 10394 |
| 192.168.1.138 | microsoft-ds (445/tcp) | 10394 |
| 192.168.1.13  | netbios-ns (137/udp)   | 10150 |
| 192.168.1.243 | netbios-ns (137/udp)   | 10150 |
| 192.168.1.4   | microsoft-ds (445/tcp) | 11011 |
| 192.168.1.13  | microsoft-ds (445/tcp) | 11011 |
| 192.168.1.138 | netbios-ns (137/udp)   | 10150 |
| 192.168.1.138 | microsoft-ds (445/tcp) | 11011 |
| 192.168.1.243 | microsoft-ds (445/tcp) | 11011 |
+-----+-----+-----+
```

334 Chapter 10 • Enterprise Scanning

Even though the example is simplistic, you can see the benefit you receive from using the database as your information-generating tool. Scanning the hosts from the DMZ revealed that all our firewall was doing was blocking port 139, which as you can see is insufficient in blocking most vulnerabilities.

There are other means of inserting data into a database; specifically, NessusWX (available from <http://nessuswx.nessus.org>), Nessus' Windows-based client. NessusWX can generate SQL statements that you can paste into your SQL server, or directly insert entries into a MySQL-based database. We find NessusWX's capability to use a SQL server better than storing the results in a file, but as the data cannot be manipulated before it is entered, we prefer using our own scripts to insert data into the database.

Differential Reporting

Differential reporting and the trend analysis information you can generate from it allows you to easily spot changes on your network. Differential reporting allows you to compare one scan against another, and receive only the changes caused when a new vulnerability has been discovered, a new host is added to the network, or a new service is operating. This allows you to reduce the size of the report you receive, and the overload generated due to it, while still allowing you to be successfully alerted to new security problems on your network.

You can generate differential reports using either of two methods: scan normally and use a database to generate differential data, or use Nessus' built-in differential reporting features through the configuration. Choosing the latter only requires marking the scan as differential, by either editing the configuration file used by Nessus or setting the **save_knowledge_base**, **kb_restore**, and **diff_scan** entries to **yes**. Alternatively, you can change these values via the GUI by activating the **KB | Enable KB saving, Reuse the knowledge bases about the host for the test**, and **Only show differences with the previous scan** options.

Once the scan has completed, the results you receive are only those new vulnerabilities, ports, or hosts that weren't present in the previous scan. The shortcoming from this type of scan is that you cannot generate any additional information from the results; for example, what was fixed, what vulnerabilities are still present although found a certain timeframe ago, and so forth. Another disadvantage of this type of differential data is that it is hard to determine what has changed. Some plugins return dynamic data, such as the plugin that retrieves the SMTP banner. Since the banner might contain the current time, it will be dif-

ferent between every two scans you conduct and will constantly appear in the differential results, even though nothing actually changed with regard to that vulnerability.

You can partially minimize this effect by excluding the information-gathering plugins from being launched again against the hosts by choosing **KB | Do not execute info gathering plugins that have already been executed**, or by setting the **kb_dont_replay_info_gathering** entry in the configuration file to **yes**. However, this would defeat the purpose of scanning the computer for all possible vulnerabilities, including any information-gathering plugins (consider, for example, a situation where an information-gathering plugin was enhanced—you will never get to see these enhancements since the new version will never run). Furthermore, it risks keeping outdated information in the Knowledge Base.

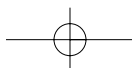
NOTE

By default, Nessus keeps records in the Knowledge Base for a period of 10 days. If you are interested in waiting more than that between differential scans, change the value of **Max age of saved KB (in secs)** found under **KB** tab, or modify the value of **kb_max_age** from 864000, which equals 10 days, to any other value.

Now that you know how to use Nessus' built-in feature for differential scanning, we'll look at how to use your database to generate similar and better differential information from the same scans. In the previous section, we implemented the table structure. In this section, we'll take things a step further by modifying the table to include an additional column to accommodate our differential scan data. This column, that we'll label **ScanDate**, will enable us to set the date at which the report is generated. We'll do this by implementing the following SQL statement:

```
ALTER TABLE ScanResults ADD ScanDate DATE;
```

In addition, we will need to modify the **nbeparser.pl** script, from the previous section, so that it includes the **ScanDate** column. The value to which we will set the **ScanDate** will be the reserved word **NOW()**. MySQL will, in turn, convert this to the date and time at which an entry was inserted. This change to the script will result in:



336 Chapter 10 • Enterprise Scanning

```
my $SQL = "INSERT INTO ScanResults SET IP='".$values[0]."',
Port='".$values[1]."', PluginID='".$values[2]."', Type='".$values[3]."',
Description=".$dbh->quote($values[4]).", ScanDate=NOW()";
```

Now, we'll generate two separate scans. The first scan will be conducted today, without making any changes. Next, we will alter our network so that it appears that one of our hosts is no longer responding. Additionally, we will turn off a few of our services on another one of our hosts. This will enable us to create a second scan that will now appear as if it is being completed one day after our first scan. Once we've imported results from both scans into our database, we can start to derive meaning from the resulting data. From here on, we will assume your database contains results for scans that were completed for the same range of IP addresses.

Let's begin by pulling any scan dates with which we can work. This can be easily accomplished by using the following SQL statement:

```
select DISTINCT(ScanDate) from ScanResults;
```

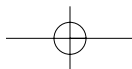
```
+-----+
| ScanDate |
+-----+
| 2004-06-15 |
| 2004-06-16 |
+-----+
```

Now, we can provide a list of hosts that were discovered during the scan and a list of hosts that were absent using the following SQL statement:

```
SELECT DISTINCT(CurrentScan.IP) FROM ScanResults AS CurrentScan LEFT JOIN
ScanResults ON (CurrentScan.IP = ScanResults.IP AND CurrentScan.ScanDate !=
ScanResults.ScanDate AND ScanResults.ScanDate = '2004-06-15') WHERE
CurrentScan.ScanDate = '2004-06-16' AND ScanResults.IP IS NULL;
Empty set (0.00 sec)
```

```
SELECT DISTINCT(CurrentScan.IP) FROM ScanResults AS CurrentScan LEFT JOIN
ScanResults ON (CurrentScan.IP = ScanResults.IP AND CurrentScan.ScanDate !=
ScanResults.ScanDate AND ScanResults.ScanDate = '2004-06-16') WHERE
CurrentScan.ScanDate = '2004-06-15' AND ScanResults.IP IS NULL;
```

```
+-----+
| IP      |
+-----+
| 192.168.1.52 |
+-----+
```



The first statement returns a list of the new hosts that appeared on the network, and the second statement returns a list of IPs that have dropped off the network. As you can see, host 192.168.1.52 is absent from the second scan's results.

We can now create similar lists for both ports and vulnerabilities. Let's start by using the SQL statement that will retrieve the new ports that differed between our two scans:

```
SELECT DISTINCT(CurrentScan.Port), CurrentScan.IP FROM ScanResults AS
CurrentScan LEFT JOIN ScanResults ON (CurrentScan.Port = ScanResults.Port
AND CurrentScan.ScanDate != ScanResults.ScanDate AND ScanResults.ScanDate =
'2004-06-15') WHERE CurrentScan.ScanDate = '2004-06-16' AND
ScanResults.Port IS NULL;
```

Empty set (0.05 sec)

```
SELECT DISTINCT(CurrentScan.Port), CurrentScan.IP FROM ScanResults AS
CurrentScan LEFT JOIN ScanResults ON (CurrentScan.Port = ScanResults.Port
AND CurrentScan.ScanDate != ScanResults.ScanDate AND ScanResults.ScanDate =
'2004-06-16') WHERE CurrentScan.ScanDate = '2004-06-15' AND
ScanResults.Port IS NULL;
```

```
+-----+-----+
| Port                | IP                |
+-----+-----+
| mysql (3306/tcp)    | 192.168.1.243    |
| x11 (6000/tcp)     | 192.168.1.52     |
| loc-srv (135/udp)  | 192.168.1.243    |
| ssh (22/tcp)       | 192.168.1.52     |
| unknown (1029/udp) | 192.168.1.243    |
| ftp (21/tcp)       | 192.168.1.243    |
+-----+-----+
```

Here it's easy to spot that host 192.168.1.243, which is present in both scans, had some ports from the first scan close before our second scan was run. Additionally, we can see that host 192.168.1.52 had a port open in the time period between the scans. Moreover, this host is absent from our second scan. However, these two facts are not equal in importance. Simply put, if host 192.168.1.52 is no longer there, its open and closed ports are meaningless to us. Once we recognize this, we can easily filter out host 192.168.1.52 from our results by creating a more complex SQL statement:

338 Chapter 10 • Enterprise Scanning

```
SELECT DISTINCT(CurrentScan.Port), CurrentScan.IP FROM ScanResults AS
CurrentScan LEFT JOIN ScanResults ON (CurrentScan.Port = ScanResults.Port
AND CurrentScan.ScanDate != ScanResults.ScanDate AND ScanResults.ScanDate =
'2004-06-16') WHERE CurrentScan.ScanDate = '2004-06-15' AND CurrentScan.IP
NOT IN ('192.168.1.52') AND ScanResults.IP IS NULL;
```

```
+-----+-----+
| Port          | IP          |
+-----+-----+
| mysql (3306/tcp) | 192.168.1.243 |
| loc-srv (135/udp) | 192.168.1.243 |
| unknown (1029/udp) | 192.168.1.243 |
| ftp (21/tcp)      | 192.168.1.243 |
+-----+-----+
```

In the **NOT IN** clause, we place any IPs that previously appeared as being absent from the previous scan, thus filtering them out from the information in which we are interested. A very similar SQL query can show the vulnerabilities that have been addressed:

```
SELECT DISTINCT(CurrentScan.PluginID), CurrentScan.IP, CurrentScan.Port
FROM ScanResults AS CurrentScan LEFT JOIN ScanResults ON
(CurrentScan.PluginID = ScanResults.PluginID AND CurrentScan.ScanDate !=
ScanResults.ScanDate AND ScanResults.ScanDate = '2004-06-15') WHERE
CurrentScan.ScanDate = '2004-06-16' AND CurrentScan.IP NOT IN
('192.168.1.52') AND ScanResults.PluginID IS NULL;
```

```
+-----+-----+-----+
| PluginID | IP          | Port          |
+-----+-----+-----+
| 11580 | 192.168.1.254 | general/udp |
+-----+-----+-----+
```

```
SELECT DISTINCT(CurrentScan.PluginID), CurrentScan.IP, CurrentScan.Port
FROM ScanResults AS CurrentScan LEFT JOIN ScanResults ON
(CurrentScan.PluginID = ScanResults.PluginID AND CurrentScan.ScanDate !=
ScanResults.ScanDate AND ScanResults.ScanDate = '2004-06-16') WHERE
CurrentScan.ScanDate = '2004-06-15' AND CurrentScan.IP NOT IN
('192.168.1.52') AND ScanResults.PluginID IS NULL;
```

```
+-----+-----+-----+
| PluginID | IP          | Port          |
+-----+-----+-----+
```

```

| 10916 | 192.168.1.243 | microsoft-ds (445/tcp) |
| 12054 | 192.168.1.243 | microsoft-ds (445/tcp) |
| 10915 | 192.168.1.243 | microsoft-ds (445/tcp) |
| 12209 | 192.168.1.243 | microsoft-ds (445/tcp) |
| 10481 | 192.168.1.243 | mysql (3306/tcp) |
| 10914 | 192.168.1.243 | microsoft-ds (445/tcp) |
| 11890 | 192.168.1.243 | loc-srv (135/udp) |
| 10913 | 192.168.1.243 | microsoft-ds (445/tcp) |
| 10092 | 192.168.1.243 | ftp (21/tcp) |
+-----+-----+-----+

```

With these two statements, you can spot which new vulnerabilities appeared (in this case, a vulnerability whose script ID is 11580) and which vulnerabilities appear to have been addressed (in this case, vulnerabilities with script IDs 10916, 12054, 10915, 12209, 10481, 10914, 11890, 10913 and 10092).

This wealth of information can be very useful for an administrator when handling a large network, as some vulnerabilities might take longer to fix, in which case seeing them appear and reappear in every report you receive might degrade your ability to understand this as well as your ability to follow the reports as a whole. In contrast, showing you only the changes between the two scans, while giving you the ability to generate a complete report at will, gives you the best of both worlds, seeing the data both in full and differential formats.

Another feature that we can incorporate using simple SQL queries is the ability to see vulnerability trends, or how long a certain vulnerability has been around on the server in question, as data can be easily fetched by its scan date. To better illustrate this, we added some data into the database, after starting new services on the machine 192.168.1.243.

We start by requesting a list of all the unique PluginIDs we can find in the table:

```

SELECT DISTINCT(PluginID) FROM ScanResults
+-----+
| PluginID |
+-----+
| 11834 |
| 12264 |
| 10287 |
| 10916 |

```

340 Chapter 10 • Enterprise Scanning

```

| 10342 |
| 12054 |
| 11835 |
|.....|
| 11580 |
| 10719 |
+-----+

```

Then we use an SQL statement that will show for a particular PluginID and an IP, how long that vulnerability has been present:

```
SELECT IP, Port, ScanDate FROM ScanResults WHERE PluginID='10481' ORDER BY
IP, Port;
```

```

+-----+-----+-----+
| IP          | Port          | ScanDate     |
+-----+-----+-----+
| 192.168.1.243 | mysql (3306/tcp) | 2004-06-15 |
| 192.168.1.243 | mysql (3306/tcp) | 2004-06-17 |
+-----+-----+-----+

```

We specifically chose a vulnerability that was there, disappeared, and reappeared. As you can see, this vulnerability appeared to have been fixed, but resurfaced later—a very valuable piece of information. Why has this vulnerability resurfaced? Was some kind of firewall rule changed? Was this vulnerability only partially addressed? This information would be hard to come by if you didn't use this type of differential information gathering.

NOTE

You can use, as we did in this query, the **ORDER BY** directive of SQL to prevent the same vulnerability from appearing on the same IP, but on different ports, from confusing the trend the SQL statement returns.

We now want to generate some graphs from the data we collected. We'll do this by using MySQL's capability to generate on-the-fly comma-separated value (CSV) files and the SQL statement's power to return the number of entries it has for a specific row. The following SQL statements will return for each risk factor

the number of entries divided by the different scan dates. We also added the raw data returned by each of the SQL statements:

```
SELECT ScanDate, COUNT(ScanDate) FROM ScanResults GROUP BY ScanDate INTO
OUTFILE 'vuln.total.data';
```

```
+-----+-----+
| ScanDate | COUNT(ScanDate) |
+-----+-----+
| 2004-06-15 |          77 |
| 2004-06-16 |          55 |
| 2004-06-17 |          54 |
+-----+-----+
```

```
SELECT ScanDate, COUNT(ScanDate) FROM ScanResults WHERE Description LIKE
'%Risk Factor : High%' OR Description LIKE '%Risk Factor : Serious%' GROUP
BY ScanDate INTO OUTFILE 'vuln.high.data';
```

```
+-----+-----+
| ScanDate | COUNT(ScanDate) |
+-----+-----+
| 2004-06-15 |          10 |
| 2004-06-16 |           6 |
| 2004-06-17 |           6 |
+-----+-----+
```

```
SELECT ScanDate, COUNT(ScanDate) FROM ScanResults WHERE Description LIKE
'%Risk Factor : Medium%' GROUP BY ScanDate INTO OUTFILE 'vuln.medium.data';
```

```
+-----+-----+
| ScanDate | COUNT(ScanDate) |
+-----+-----+
| 2004-06-15 |          10 |
| 2004-06-16 |           7 |
| 2004-06-17 |           7 |
+-----+-----+
```

```
SELECT ScanDate, COUNT(ScanDate) FROM ScanResults WHERE Description LIKE
'%Risk Factor : Low%' GROUP BY ScanDate INTO OUTFILE 'vuln.low.data';
```

```
+-----+-----+
| ScanDate | COUNT(ScanDate) |
+-----+-----+
```

342 Chapter 10 • Enterprise Scanning

```

+-----+-----+
| 2004-06-15 |           25 |
| 2004-06-16 |           21 |
| 2004-06-17 |           18 |
+-----+-----+

```

All you need to do now to generate a nice graph from this data is to use the following gnuplot script:

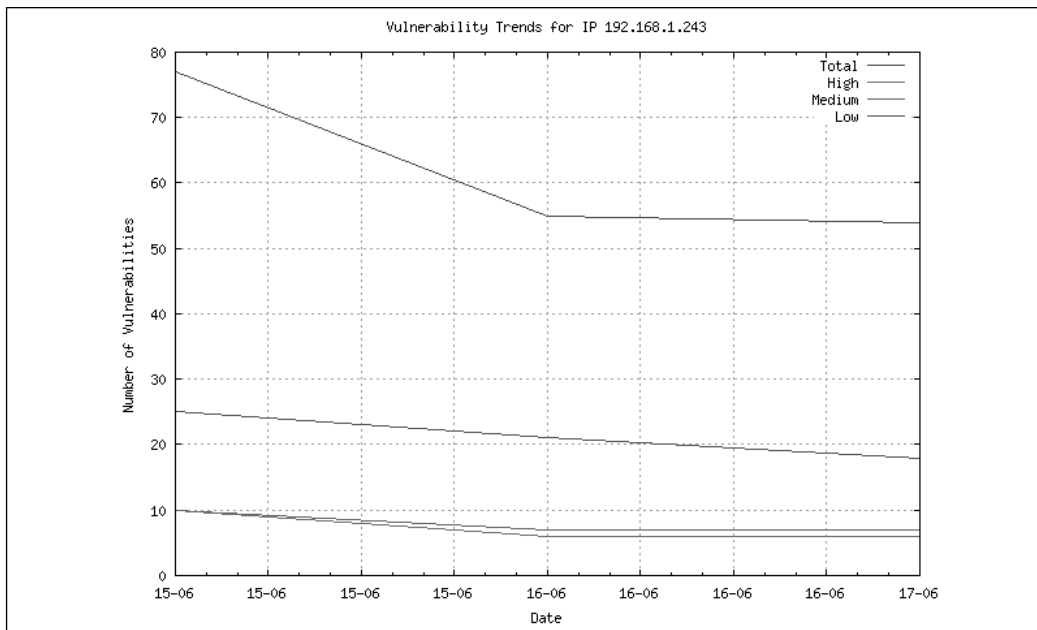
```

set term png small
set data style lines
set grid
set yrange [ -10 : ]
set title "Vulnerability Trends for IP 192.168.1.243"
set xlabel "Date"
set ylabel "Number of Vulnerabilities"
set xdata time
set timefmt "%Y-%m-%d"
set xrange ["2004-06-15":"2004-06-17"]
set format x "%d-%m"
set yrange [0:]
plot "vuln.total.data" using 1:($2) title "Total"\
    , "vuln.high.data" using 1:($2) title "High"\
    , "vuln.medium.data" using 1:($2) title "Medium"\
    , "vuln.low.data" using 1:($2) title "Low"

```

The utility will generate a graph similar to what is shown in Figure 10.8.

Figure 10.8 Trend for 192.168.1.243



The more extensive data you have, the more interesting and useful this graph becomes. Taken over time, this graph illustrates how vulnerabilities are progressing for this particular host. Continuing our data mining, we would now like to count the number of vulnerabilities divided by dates and IPs by issuing the following SQL statement:

```
SELECT ScanDate, IP, COUNT(IP) FROM ScanResults GROUP BY IP, ScanDate;
```

```
+-----+-----+-----+
| ScanDate | IP | COUNT(IP) |
+-----+-----+-----+
| 2004-06-15 | 192.168.1.243 | 31 |
| 2004-06-16 | 192.168.1.243 | 15 |
| 2004-06-17 | 192.168.1.243 | 17 |
| 2004-06-15 | 192.168.1.254 | 18 |
| 2004-06-16 | 192.168.1.254 | 19 |
| 2004-06-17 | 192.168.1.254 | 17 |
| 2004-06-15 | 192.168.1.4 | 21 |
| 2004-06-16 | 192.168.1.4 | 21 |
| 2004-06-17 | 192.168.1.4 | 20 |
```

344 Chapter 10 • Enterprise Scanning

```
| 2004-06-15 | 192.168.1.52 |          7 |
+-----+-----+-----+
```

The only piece of information we are lacking is those vulnerabilities that are persistent. A vulnerability can be declared persistent if it persists for a long time and is present in every scan. Thinking graphically, we mean that the vulnerability's line starts on one end of our graph and proceeds to the other without segmentation. By performing the following query, we can get a perspective for each plugin we tested for, when it was first detected, and how many hosts were affected:

```
SELECT ScanDate, COUNT(ScanDate), PluginID, Port FROM ScanResults GROUP BY
ScanDate, PluginID, Port ORDER BY PluginID, ScanDate;
```

```
+-----+-----+-----+-----+
| ScanDate | COUNT(ScanDate) | PluginID | Port |
+-----+-----+-----+-----+
| 2004-06-15 |          1 | 10028 | domain (53/tcp) |
| 2004-06-16 |          1 | 10028 | domain (53/tcp) |
| 2004-06-17 |          1 | 10028 | domain (53/tcp) |
| 2004-06-15 |          1 | 10092 | ftp (21/tcp) |
| 2004-06-17 |          1 | 10092 | ftp (21/tcp) |
|-----|-----|-----|-----|
| 2004-06-16 |          2 | 12053 | general/tcp |
| 2004-06-17 |          2 | 12053 | general/tcp |
| 2004-06-15 |          1 | 12054 | microsoft-ds (445/tcp) |
| 2004-06-15 |          1 | 12209 | microsoft-ds (445/tcp) |
| 2004-06-15 |          3 | 12264 | general/icmp |
| 2004-06-16 |          3 | 12264 | general/icmp |
| 2004-06-17 |          3 | 12264 | general/icmp |
+-----+-----+-----+-----+
```

You can easily spot in this query output what vulnerabilities appear more than others, and what vulnerabilities are particular to certain hosts.

To summarize, as the data is in a relational database, you can generate a multitude of SQL statements to go through this data and analyze it. The data can be further extended to contain such columns as *assignee*, the person in charge of fixing a particular vulnerability, *ignore* allowing you to ignore certain results from

future reports, and so forth. We explain the *ignore* column in better detail in the next section.

Filtering Reports

The previous section focused on the benefits of differential reporting and how to generate differential reports. In this section, we look at how to filter out wrong or misleading content from your reports. This will enable you to filter out enterprise-wide false positives and irrelevant vulnerabilities, generate reports containing less information than the original database, and divide the reports based on the type of operating system. In other words, you will be able to locate the relevant data that you seek.

As before, we'll be using a MySQL database containing the single table with its differential data that we created in the previous section, and the Perl script that we used to insert the data into the table. We will start by generating a table of IPs vs. operating systems with a simple SQL statement:

```
SELECT IP, MID(Description, LENGTH('The remote host is running')+1,
LOCATE(';', Description)-LENGTH('The remote host is running')-1) AS OS
FROM ScanResults WHERE PluginID='11936' GROUP BY IP, OS;
```

```
+-----+-----+
| IP           | OS                               |
+-----+-----+
| 192.168.1.243 | Microsoft Windows 2000 Server |
| 192.168.1.4   | Microsoft Windows XP           |
+-----+-----+
```

This SQL statement looks through the database for all the records of the Nessus plugin *OS Identification*. When such entries are found, they are grouped together. The result from the *Description* column is made a bit more readable by using MySQL's string handling functions.

NOTE

OS detection algorithms, whether they are ICMP based or use Nmap's fingerprinting technique, are rarely accurate or consistent. Therefore, if you want to better generate reports divided by OS types, we suggest that you build an additional table where you can enter and maintain IP vs. OS information manually.

346 Chapter 10 • Enterprise Scanning

You can also limit your data to a certain IP or IP range. As our table contains numerous records, we will need to eliminate all extraneous data, fix the most critical issues, and then handle those issues that remain. We can use simple queries to filter out IPs, by using *IP='192.168.1.254'* as our query string. Alternatively, we can use a range of IPs by using *IP IN ('192.168.1.52','192.168.1.254')*. Furthermore, we can create a subset of the network by using an *IP LIKE '192.168.1.%'*.

At times, you might need to scan a large part of the network, but not the entire network. In such an instance you can use Nmap's capability to parse the value of an IP/netmask pair into a list of IPs: *nmap -sL 192.168.1.1/29*. This will create a list of IPs that fall under that range. The same can be done for 192.168.1-2.1-2, which will return the list 192.168.1.1, 192.168.1.2, 192.168.2.1, 192.168.2.2. This list can then be taken to the SQL queries *WHERE* section, and provided as is.

To save time, use the table from Table 10.1 and insert it into the MySQL table. Be sure to select the relevant IPs by asset instead of only by the IP. Now, be sure to use the following table structure for your records:

```
CREATE TABLE AssetList (  
    ID INT AUTO_INCREMENT,  
    AssetName TEXT,  
    IPs TEXT,  
    ExpectedResult TEXT,  
    Contact TEXT,  
    Frequency TEXT,  
    PRIMARY KEY ID (ID)  
);
```

After entering the details into the table, use the IP addresses found in the *IPs* column to filter out the data in the ScanResults table.

Most enterprise networks are bound to have a few irrelevant vulnerabilities. Nessus was built to report everything, and then let someone else sort through the results. That way, it's possible for you to define what is important and what is not. This makes it possible for you to be sure that you do not miss any vulnerabilities, because Nessus will not ignore any. However, when you are dealing with a large-scale network, this can result in a few hundred irrelevant vulnerabilities. For example, items like administrative privilege vulnerabilities will appear when

Nessus is provided with the administrative credentials required to remotely access the workstations' registry.

As all our data is in the database, we can quickly weed out these records by explicitly deleting the records from our database where a certain PluginID is equal to a certain number, or by adding a new column to the ScanResults table that marks the record as irrelevant. The second option enables you to reactivate those entries in the future as changes occur in the corporate policy regarding that vulnerability.

Additionally, some vulnerabilities are simply false positives, caused by a variety of issues. Unfortunately, fixing these false positives requires time and effort. The quickest fix requires a familiarity with Nessus' scripting language and the ability to find and repair the underlying problem. Therefore, we must adopt an approach similar to the solution we used with the irrelevant vulnerabilities described in the previous two paragraphs. To do this, we must mark those vulnerabilities that are false positive as irrelevant. We will also need to locate and fix the root cause of the false positive, by e-mailing bugs@nessus.org, reporting it online at <http://bugs.nessus.org/>, or e-mailing the Nessus mailing list (available at <http://list.nessus.org>). Alternatively, you can fix the offending NASL on your own, or contact the NASL's author—a contact e-mail address is usually available at the top of each Nessus Plugin.

Third-Party Tools

In this section, we highlight some third-party tools you can use to assist your deployment, maintenance, and usage of Nessus.

Extracting Information from a Saved Session Using sd2nbe

The Nessus daemon has the capability to store information about scanning sessions in a file for resuming interrupted scans. This option is normally used to recover sessions; however, it is not limited to that. You can take the information saved in the session and convert it to an NBE data file using the sd2nbe tool available from www.tifaware.com/perl/sd2nbe/. Once the session information is in NBE format, we can open it using the Nessus client or convert it to HTML or XML.

Nessus Integration with Perl and Net::Nessus::ScanLite

You might have thought of building a wrapper to the Nessus command-line client to automate your deployment and to automatically schedule scans. To make this easier, you can use Net::Nessus::ScanLite, a Perl module that allows you to implement your very own Nessus client. The Perl module is almost a complete command-line Nessus client, supporting all of the Nessus client features and offering an API to easily use the results of a scan.

Using simple scripts, you can connect to the Nessus daemon, grab the results, and dump them directly into the database without generating a single file. You can also extend this to create a complete Perl-based web interface where you choose what IP to scan. That same Perl-based web page can call the script that runs Nessus, and return the results. This tool is available from <http://search.cpan.org/~jpb/Net-Nessus-ScanLite-0.01/>.

Users who have trouble installing the necessary dependencies can use the automated CPAN installation, which takes care of all the dependencies required.

NOTE

As this is a book written by security professionals who are paid to be paranoid, we'll always suggest using the nonautomated installation mechanism that allows you to check PGP signatures instead. The following automated process does not check PGP signatures, and its documentation admits to weak security:

"There's no strong security layer in CPAN.pm. CPAN.pm helps you to install foreign, unmasked, unsigned code on your machine. We compare to a checksum that comes from the Net just as the distribution file itself. If somebody has managed to tamper with the distribution file, they may have as well tampered with the CHECKSUMS file. Future development will go towards strong authentication."

(Source: <http://search.cpan.org/~jhi/perl-5.8.0/lib/CPAN.pm#SECURITY>)

As this documentation reminds you, an attacker who compromises the CPAN servers can replace the Perl module that you're installing while also replacing the checksum on the same server. The only way to avoid having a vital piece of your security infrastructure compromised by an attacker's hostile code is to check that code's PGP signature before installing it. Avoid using easy methods that ignore this risk, especially on

your organization's security infrastructure. Exercise best practices to avoid this infrastructure compromise or you might find yourself losing control of the system, and your job!

Executing the following commands will install the module:

```
# perl -MCPAN -e shell
cpan> install Net::Nessus::ScanLite
```

A very easy way to manage multiple Nessus daemon installation is by using a web interface. Several such web interfaces have been written as open-source projects. One such project is the Vulnerability Scanning Cluster (VSC). The VSC interface was written in PHP and uses the MySQL database as its backend for storing information. VSC allows users to manage hosts by hierarchical order, to select different scanning policies, to schedule future and recurring scans, and to view scan results. The tool is available from www.sourceforge.net/projects/vscweb/.

Nessus NBE Report Parsing Using Parse::Nessus::NBE

If you already have several Nessus reports in NBE data format and want to process them without much hassle, you can use the Parse::Nessus::NBE module. This module handles the regular expressions work for you, leaving you to simply request different types of information within the reports. Parse::Nessus::NBE can locate banners, open ports, plugins, numbers of vulnerabilities, web directories discovered, NFS-related issues, OS types, SNMP related issues, OS type statistics, service statistics, and vulnerability statistics.

This tool is available from <http://search.cpan.org/~dkyger/Parse-Nessus-NBE-1.1/>. Again, new users can use the following automated CPAN installation to handle the required dependencies, but should remember the resulting risk to their organization's security infrastructure. Those choosing this method can execute the following commands to install:

```
# perl -MCPAN -e shell
cpan> install Parse::Nessus::NBE
```

Common Problems

Several problems might arise while you are using the Nessus daemon to scan your network. These problems are not necessarily related to bandwidth or a vulnerability, but rather to a product vendor's improper handling of abnormal and unexpected traffic being directed at their product.

In this section, we divide the problems that might surface when scanning your network with Nessus into the following categories: aggressive scanning, volatile applications, and printer carnage. Toward the end of this section, we also describe another shortcoming that might show up when running Nessus against your enterprise's workstations. These workstations might be turned off during some part of the scan or change their IPs due to a restart, and the results pertaining to these workstations need to be addressed differently.

Aggressive Scanning

The type of portscan performed on your network has an effect on the stability of numerous types of hardware and software. We need to remember that while the main goal of the portscan phase is to detect open ports, it needs to do so while making sure that the software and hardware being scanned will "survive" the portscan, so that we can detect vulnerabilities in that software/hardware and do not needlessly reduce their uptime.

The Nessus daemon uses Nmap for its portscanning back engine, and as such, it supports all of Nmap's portscanning techniques, including TCP connect(), SYN, FIN, Xmas Tree, SYN FIN, FIN SYN, and NULL. Each scan type has a different effect on the software and hardware being scanned. As we are not testing the vendor's product endurance to portscanning but rather searching for vulnerabilities, we will want to use the closest thing to a normal user's connection: TCP connect(). Other scan types can cause inadvertent DoS conditions.

NOTE

Even though TCP connect() is considered the slowest scan type, it emulates in closest resemblance what a normal user would go through to connect to a remote host's ports.

You can also configure through Nessus the speed at which Nmap tries to connect to the remote server. The speed settings range from *Paranoid*, the slowest, to *Insane*, the fastest. Faster scans will often result in inaccurate results, as Nmap fails to get a response from the target on one port before giving up to start a new test on another port. Nessus' default settings are set to *Normal* speed, which tries to run as quickly as possible without overloading the network and without missing hosts/ports. We recommend against changing Nmap's scanning speed from the Normal settings, as the benefit of scanning faster is minimal in comparison to the potential problem of not detecting a certain port as open.

The Nmap settings that have a more visible effect on the portscanning speed, while not compromising the accuracy, are *Min RTT Timeout*, *Max RTT Timeout*, and *Initial RTT Timeout*. These three settings tell Nmap how long to wait for a response. The *Initial RTT Timeout* sets how long Nmap will wait for the first packets to return—there is no need to set this value any higher than the network's current maximum latency (the time it takes an ICMP packet to return from that host). The *Min/Max RTT Timeout* tells Nmap the range of the roundtrip time it can expect from the network. Limiting the range too much will cause Nmap to lose or discard packets, while setting it to a wide range will cause Nmap to scan more slowly.

We recommend setting the *Initial RTT Timeout* to 1000 (milliseconds) for LAN networks and to 3000 (milliseconds) for WAN networks, and setting the *Min* and *Max RTT Timeout* to .5 the *Initial* for the *Min* and 1.5 times for the *Max*.

One last Nmap setting that has an effect on the speed of the scan is *Host Timeout*. By setting this parameter, you are telling Nmap when to “give up” on a certain host and consider the current open-port list final. Nmap will consider any packet that was dropped as a lost packet and not as a closed port, and as such, when a firewall or firewalled host is scanned, it will take longer to return open port results. Nmap has no default timeout setting, and will continue to scan the firewall several times before giving up. The *Host Timeout* setting allows you to define this timeout interval, and prevent Nmap from spending several hours on the same host for just the portscanning phase.

We recommend setting the *Host Timeout* to 21,600,000 (milliseconds); that is, roughly 6 hours. However, if your network is fast, and you have no need to scan firewalls or hosts behind a firewall, you can lower this setting to achieve a faster scan. Remember, don't lower it too much, as portscanning takes time.

Volatile Applications

Not all of the possible problems stem from portscanning alone. Many applications are easily affected by Nessus' interaction on the ports on which they listen. Application developers expect the program connecting to it to behave in a certain way, according to the predefined protocol. A web server developer may assume that the client is always a web browser; an FTP server developer is likely to consider an FTP client to make incoming connections. This means that the QA process of the program is likely to run a client that behaves in a normal way, and doesn't send unexpected content.

The Nessus plugins are far from perfect in this respect. They were built to detect vulnerabilities. A vulnerability stems from the combination of a misbehavior on the part of the client and the server's poor reaction to that misbehavior. As such, Nessus plugins are likely to misbehave from the server's point of view. In addition, many of the Nessus plugins do not just assume a vulnerability exists because of the remote server type and version, but rather actually try to probe more deeply, usually attempting to exploit the problem.

By attempting to exploit vulnerabilities, Nessus tends to create fewer false positives, as a system that isn't vulnerable won't be successfully exploited. This is more accurate simply because it simulates the attacker's interaction with a vulnerability. Checking for vulnerabilities by attempting their exploitation also leads to detecting vulnerabilities in products not previously known to be vulnerable. One such example is the infamous AUX/LPT/COM1 DoS vulnerability. This vulnerability stems from the fact that when you request one of these reserved names as a filename from a remote web server, the server's attempt to access that filename will cause it to hang and subsequently disrupt the web server's service to clients. As this vulnerability affects more than a single server, the same plugin that detects it on Vendor X's product can detect it on Vendor Y's product as well, as the attack works for both products in a similar manner.

This behavior, however, affects a wider range of products than that which the Nessus plugin was built to detect, causing problems when you scan products that are relatively obscure, went untested by security analysts, weren't properly upgraded to prevent such problems, and so forth. These machines might crash when scanned or their applications might stop responding, requiring a restart of the affected applications or even the entire server.

Another issue that can arise from badly written products is the effect a certain plugin might have on the tested machine when not conforming to the

vendor's specified protocol. From the point of view of the person who developed an application, anyone connecting to his or her port is trying to access the developer's service and thus knows what product is listening on that port, and as such will use the proper syntax. This isn't necessarily the case when a vulnerability scan is communicating with that port, especially in those cases where the scanner doesn't even know what application is listening. Before you disregard this, think about how many different web servers exist, from mainstream servers to small server programs on network printers that don't even respond to every HTTP command.

A badly written protocol implementation might be waiting for a *QUIT* command, and if such a command does not arrive, it might fail to close the connection (such as is the case with some SMTP servers). Due to this, Nessus' plugins launched against that port that fail to send the *QUIT* command will create a new connection that the product will never close. Over the course of multiple plugins that interact with the product's port, the scan will cause the product to create multiple sockets, consuming valuable resources and overloading the machine it is run on, and also possibly causing the failure of the server altogether.

This issue can also manifest itself during the portscan, as no data is sent to those ports. A badly written application might try to read a set size of bytes from the socket, and as no data is sent, the product will freeze waiting for more bytes or otherwise fail.

In most of the cases we mention here, where the product fails, the behavior is in fact a problem in the product and not in the Nessus scanner, and the product's failure to handle unexpected input without failure should itself be addressed as a vulnerability. For example, older Cisco routers might still hang when receiving Nmap's UDP scan packet on their syslog port, UDP 514. However, such unexpected behavior may end up causing the program to fail without your knowledge of the cause. Therefore, it is important to make sure that the majority of your critical products are properly tested for robustness to portscanning and testing during the testing lab phase discussed earlier. One of this book's authors used Nmap to scan a Cisco router and caused it to hang so badly that it lost its configuration, requiring a router administrator to enter the configuration line by line at the router's console. All this from a single UDP 514 packet! This type of failure could have been prevented with a router firmware patch deployed proactively and served as a DoS vulnerability, but it disconnected a network for 30 minutes. This problem should probably be found in a lab, instead of on the production network.

354 Chapter 10 • Enterprise Scanning

Another issue that might occur due to Nessus launching plugins against a specific product is that the product might not be designed to receive numerous simultaneous connections. You need to remember that Nessus makes scans run faster by launching more than a single plugin against each host. Badly written products may not support such multiple connections, and may as a result fail to address new connections or even crash. This type of behavior is difficult to predict and might occur even if you have done previous tests in a test lab, as the problem may require particular circumstances only likely in production use, such as high CPU load or network utilization.

Printer Problems

One of the most common problems occurs when scanning network appliances, especially printers. Scanning a printer can waste money in a form that isn't easily replaceable or recoverable. Some printers using old firmware versions handle portscans amazingly badly—they begin to print huge amounts of garbage, wasting large amounts of paper and ink or toner. This is because the printer wastefully thinks that the scan is someone trying to print something.

Printers aren't the only piece of network appliance affected by such scans. Voice over IP (VoIP) products, billing devices, automated teller machines (ATMs), time clock machines, parking systems, and basically any other product that answers to IP traffic but wasn't coded to handle unexpected input well or tested for security vulnerabilities by the manufacturer may display erratic behavior during the scan. The effects a scan would have on such devices aren't always wasteful as in the case of a printer, but going through your corporate building and restarting all the time clock machines is something you will want to avoid.

NOTE

Think of the situation in which your scan brings down all the time clock machines in your organization during the night, and you are violently awakened at 6:00 A.M., after the shift changed and find out that no one can clock in. Not only will this cause havoc in the corporation, it will also get you out of bed earlier than expected.

At least in the case of printers, the Nessus daemon now incorporates a test to specifically detect whether the IP being scanned is a printer, and, if this is the case, prevent the scan from testing that IP's printing-related ports. Still, we suggest that you avoid testing network devices whenever possible, unless you can verify that scanning these devices doesn't cause them to misbehave.

Scanning Workstations

Scanning workstations is very tricky. Workstations are restarted frequently, and might not be available during scanning hours, especially when scans occur in the middle of the night. In addition, scanning workstations might affect their reaction speed; the workstation might change its IP address due to Dynamic Host Configuration Protocol (DHCP) usage, and its vulnerabilities should usually be rated differently from servers' vulnerabilities.

We still believe that scanning workstations is an essential part of your organization's way of addressing the security issues on your network. Despite the inherent challenges, we recommend that you put forth the extra effort and scan workstations as part of your regular network security scanning.

If the corporate servers are secure, but you have over 10,000 workstations wide open to attack, your network cannot be considered secure. A single worm entering the internal network can affect these workstations and thus the rest of the network, by disturbing them due to the worm exploiting the vulnerability and as an effect bringing the network down due to the resultant excessive worm traffic. This is no imaginary scenario: MSBlaster and Sasser are just recent examples of worms that affected both corporate servers and workstations. Networks that made sure their servers were secure were still affected by the worm attacking their workstations.

The issue isn't just limited to these worms that affect only Windows-based machines. UNIX machines have been attacked by a small number of worms as well. Additionally, worms that attack IIS, Apache, OpenSSL, Microsoft SQL Server, and others will affect development networks as well. A worm attacking your development network still affects your corporate servers, where the loss of revenue occurs immediately when a developer is unable to update a production server application because his workstation has to be taken offline for worm removal.

NOTE

Another reason why development machines are so important to scan is that they often run experimental versions of operating systems or applications, typically without recent service packs, as those machines get reinstalled frequently. You should make sure those machines are protected from the rest of the network if they cannot be secured.

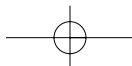
One of the problems mentioned previously is that workstations might not be available for scanning. This is especially true for employees who work in shifts, or turn off their workstations once they leave the office. For these workstations, scanning during off-peak hours is futile, as the machines cannot be scanned when they're off. However, these workstations are no less critical than those computers that are kept constantly on.

NOTE

You have to remember that the weakest link in your network's security is the computer that an attacker can compromise most easily. The CEO's secretary's computer is sometimes as valuable as the CEO's computer in terms of intellectual property. Additionally, most firewall policies grant much more privilege to an arbitrary internal computer than they do to outside computers. Attackers who compromise a single internal machine and use it to attack others are in much better shape than when they had no internal access.

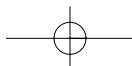
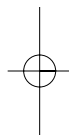
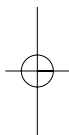
Another problem is that workstations tend to have a weaker CPU and less memory compared to servers. As such, they will be more affected by Nessus' scans. The Nessus scan on those hosts will cause them to consume more resources than usual. As this cannot be avoided, we suggest considering running the workstations scans at off-peak hours, while keeping in mind that you need to instruct the employees to avoid turning off their computers when they leave.

Another problem that rises from large-scale corporate networks is the frequent use of DHCP. The protocol is used in most corporations to avoid the need to statically assign IP addresses to each workstation on the network. This in turn causes a problem for someone trying to understand what host was vulnerable at



the time of the scan, as the IP it was allocated might be now, at the time of reading the vulnerability report, assigned to a different computer. This problem can be easily avoided by instructing the Nessus daemon to report vulnerabilities by the MAC address, or Ethernet card hardware address, of the vulnerable computer instead of by its IP address. This feature can be activated via the configuration file's **use_mac_addr** entry.

The last point to consider about workstation scanning is that not all vulnerabilities can be measured in the same way as when you're scanning a corporate server. A workstation wide open for file sharing might not be as critical as a corporate server having the same vulnerability. This is because a workstation's stored information is usually less valuable than that of the corporate server, and this is true in most cases where a centralized server stores all the information (documents, spreadsheets, and so forth). The problem is much harder to solve, and requires better planning of what tests you want to perform on the workstation and what data you want to filter out from the reports prior to going over them.



Summary

Once a security officer concludes that enterprise scanning is needed, he or she is usually baffled by the big questions: “How do I do it? And how much effort must I put into that?” These two questions are answered in detail in this Enterprise Scanning chapter. We all know that planning your deployment is important, and this is the case with Nessus.

Nessus requires preparation of your network for the bandwidth requirements of the scanner. Measuring these requirements is not always easy, but with a few tricks and the right third-party tools, you can measure these requirements, and understand the effect they will have on the network.

Bandwidth utilization is greatly affected by the different types of topologies you use to deploy the scanning servers. The different topologies also affect the hardware requirements and the necessary preparations for the day when you will need to scan a specific vulnerability instead of using the complete arsenal of vulnerability tests at your disposal.

As simply scanning your network is not enough, you need to place these results in some centralized location and start sorting out the relevant data. Once the data is placed in a database, we can use it to correlate the different results provided by the differential exposure to vulnerabilities from multiple locations throughout an organization. We can also use the database to see how the vulnerabilities have progressed over time. Most importantly, once the data has been placed in a single location, we can filter out any false positives and irrelevant vulnerabilities.

You are not obliged to do everything from scratch—several third-party tools exist that you can use to ease your usage of Nessus in the organization. These tools are free to use and you can extend them to further suit your needs. As with everything in life, there are several common problems with running Nessus in your organization. Some problems can be easily avoided, while others can be detected beforehand in the test lab so that their impact can be minimized.

Solutions Fast Track

Planning a Deployment

- ☑ Make a list of your network’s assets, who is responsible for them, and to whom the results should be mailed.

- ☑ Invite all the network's assets owners and managers to an overview of Nessus' capabilities, and the effects they have. Give a live demonstration.
- ☑ Use a test lab to determine the network bandwidth requirements your organization can afford.
- ☑ Automate the server's process of scanning and updating.

Configuring Scanners

- ☑ Choose a topology that suits your needs.
- ☑ Buy any additional hardware you require.
- ☑ Practice scanning for a specific threat, as in the case of a critical Microsoft advisory.

Data Correlation

- ☑ Use a database instead of files to store all the results.
- ☑ Correlate the results you receive from scans to help you concentrate on the most serious vulnerabilities.
- ☑ Generate differential results from the data stored in the database.
- ☑ Generate complex results using sophisticated SQL statements.
- ☑ Filter out from the database irrelevant vulnerabilities and false positives.
- ☑ Use third-party tools to ease the use vulnerability assessment in your organization.

Common Problems

- ☑ Avoid problems caused by scanning too aggressively.
- ☑ Test relatively unknown software and hardware in a test lab to avoid unexpected problems.
- ☑ Try to avoid scanning printers to save paper resources.
- ☑ Scan your workstations during working hours to avoid illusive hosts, or instruct your employees to leave their workstations turned on for the night.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to www.syngress.com/solutions and click on the “Ask the Author” form. You will also gain access to thousands of other FAQs at ITFAQnet.com.

Q: Is it feasible to scan my entire enterprise network periodically?

A: The question is not whether you should scan your entire network, it is whether you can afford *not* to scan it. Not knowing what vulnerabilities you have in your network is certainly not going to help you secure the organization. This chapter should give you some ideas on how to manage this obviously challenging task.

Q: How often should those scans run? You discuss daily/weekly scans, but we can hardly cope with our current rate of quarterly network audits!

A: As noted earlier, the Sasser worm started spreading in just over three weeks after the vulnerability was discovered. This “window of exposure” is getting smaller and smaller all the time, and demonstrates the need to perform vulnerability scans on a weekly basis. The real question is how to handle the wealth of information you will start receiving once the scans are configured to run automatically and frequently. The answer is technological: differential reporting, data mining through databased results, and eliminating false positives, using the techniques outlined in this chapter.

Q: When dealing with other departments, I have difficulties explaining why a certain vulnerability needs to be addressed. This is especially true when dealing with knowledgeable technical people such as system administrators and programmers. How can I convince them that the problem is indeed serious and should be dealt with promptly?

A: As a person in charge of security in your company, you probably have a good understanding of why security is a concern. We found that this is not always the case with other technical people who do not deal with security issues daily. However, this approach can be reversed. Security is an interesting subject for most technical people, and in many cases system administrators or

programmers will be extremely happy to participate in a security workshop that explains the concepts and demonstrates the risks. By exposing potentially resistant staff to the subject and raising their awareness to the risks, you will increase the level of cooperation and will find it easier to communicate any security concerns. People tend to cooperate better once they have a better understanding of what the challenges are—this can usually be done without training them to be security experts... Focus on cooperating with them, educating them, and framing your discussions within both the risks and the business needs. Finally, as much as possible, understand the technology and risks yourself, so you can explain them to others realistically and convincingly.

- Q:** How scalable are the solutions presented in this chapter? Will they work in my complicated network?
- A:** This chapter is based on our own experience overseeing implementations of enterprisewide vulnerability scanning solutions. If you break the huge task of scanning your entire enterprise into smaller tasks (like we did in this chapter) and then further distribute the tasks as appropriate among the different networks you have, you will see that although not trivial to set up, once the setup is working, the benefits are tremendous. Applying techniques such as distributed scanning and report correlation make it possible to use the Nessus scanning tool across the enterprise.

