;login:

THE USENIX MAGAZINE

December 2003 • volume 28 • number 6

Focus Issue: Security

Guest Editor: Rik Farrow

inside:

USENIX

The Advanced Computing Systems Association

nessus

by Renaud Deraison

Renaud Deraison is the original author of Nessus and the manager of the Nessus project. Nessus is the world's most popular vulnerability scanner and has more than 50,000 users.

deraison@nessus.org

and Ron Gula

Ron Gula is the CTO and co-founder, with Renaud Deraison, of Tenable Network Security, which funds the Nessus project and produces enterprise security management tools, as well as a variety of commercial active and passive vulnerability scanners.

rgula@tenablesecurity.com

The basic concept of conducting a network vulnerability assessment is to find security flaws so that they can be fixed before they are exploited. Nessus is a free, powerful, and easy-to-use network vulnerability scanner. It is an open source tool that is available at *http://www.nessus.org* for most UNIX operating systems.

Like most vulnerability assessment tools, Nessus performs a network vulnerability audit by first determining which hosts are alive, which applications are present, and then which vulnerabilities may be present. It has many techniques to determine all of this information and is constantly updated with new features and new vulnerability checks. Nessus also takes a closer look into the network and can identify unneeded servers or services that can have a greater long-term impact on security.

Nessus Architecture

Nessus can be deployed in a permanent or stand-alone configuration. Nessus installations consist of one or more "Nessus daemons," which are servers that perform the actual vulnerability testing, and one or more "Nessus clients," which are the control points to launch scans and review the collected data.

For single-host Nessus installations, such as a laptop, a user would install both the Nessus client and the Nessus daemon. In order to run a vulnerability scan, the Nessus daemon needs to be started first and then the Nessus client can be launched.

The client-server architecture of Nessus also makes it possible to deploy permanent "scanners" and have multiple users share them as a resource. For example, a Nessus daemon could be installed on a secured OpenBSD server located outside of a university network. This Nessus daemon can be connected to any Nessus client (with proper credentials such as a username/password or a certificate) to perform a scan. Multiple users can perform these scans simultaneously. Each user account on a Nessus daemon can also be restricted such that they can only scan a specific range of IP addresses.

Nessus Attack Scripting Language

The heart of Nessus's power and flexibility is the Nessus Attack Scripting Language (NASL). This interpreter is yet another computer language, but this one has the advantage of being a portable language designed to do network vulnerability checks. The syntax of NASL is beyond the scope of this article, but full documentation and almost 2000 examples are available at the Nessus Web site.

In version 2 of Nessus, this language was completely rewritten to improve speed and performance. A variety of built-in functions were also added, such as the ability to evenly slice IP ranges rather than just scanning sequentially. Almost any type of network vulnerability action can be coded with NASL. Some examples include generating a specific packet for a DoS attack, completing a TCP handshake to observe a banner string, and completing an NTLM authentication to a Windows primary domain controller.

Individual NASL scripts can be run alone. This is useful for testing, or building a set of scripts which are not run by the Nessus daemon. To test a NASL script, simply run the command nasl –t <IP Address> <script name> and the script will be attempted. What is more useful is that the entire processing of NASL can be viewed with the -T option, which can trace each step of the script and record it to a specific file for analysis. This makes writing custom NASL scripts and modifying existing ones extremely easy.

All Nessus plugins are open source and can be inspected by the Nessus user community. The Nessus project also attempts to maintain each original author's copyright information.

Installing Nessus

Nessus source code, pre-built binaries, and even automated Internet installations are available from a variety of mirror sites and from *http://www.nessus.org*. All Nessus build scripts are very robust and will complain if certain packages are missing from the desired target platform for Nessus.

On most UNIX systems, the easiest (and most insecure) way to install Nessus is with the simple command

```
lynx –source http://install.nessus.org | sh
```

Though easy, this command allows an attacker who has gained control or influence over your DNS service to redirect you to another site and provide different code for you to execute.

A more secure way to retrieve Nessus is to download the nessus-installer.sh tool from the Nessus Web site. This shell archive is a complete Nessus source distribution, which also automates the build process. PGP-signed MD5 signatures are also provided at the site for verification of the distributed content.

A compile-time decision that needs to be made is whether the X Windows Nessus client is needed. Without X Windows, there is no GUI for the Nessus client. This is not a problem, since Nessus is readily configurable from the command line and also from a variety of Nessus clients (some for Microsoft Windows platforms) that are maintained outside of the Nessus project. If an X Windows GUI is desired, then the graphical toolkit (GTK) library needs to be present on your desired flavor of UNIX. Most fully loaded installations of Linux or RedHat have the required libraries already installed. There are additional directions to install and compile the Nessus client with GTK support at *http://www.nessus.org/nessus_2_0.html*.

Once Nessus is downloaded and compiled, it will prompt the user for a few post-installation configuration options. These will prompt the user to generate a unique cryptographic certificate for the Nessus daemon and to create at least one Nessus "user" for the Nessus daemon. This user account will be used by the Nessus client to log into the Nessus daemon and launch a scan.

Starting Nessus and Launching a Command Line Scan

To start the Nessus daemon, simply use the command

```
nessusd –D
```

This will invoke the Nessus daemon and leave one process running. If the Nessus daemon is performing a scan, it will fork and several dozen Nessus daemon processes may become active.

By default, the Nessus daemon will listen for Nessus client TCP connections on port 1241. Nessus's man pages also do an excellent job of describing the location of the various configuration and logfiles for the Nessus daemon.

When Nessus is installed, it comes with the most recent vulnerability checks available. In the future, new vulnerability checks will be downloadable from the Nessus project

by typing the command nessus-update-plugins. If you are behind a Web proxy, this script needs to have some internal variables edited such that it can access the Internet through your proxy. Some Nessus users choose to create a cron job to schedule a download of the updated Nessus plug-ins once per day or once per week.

The Nessus command line client has a variety of features, and there are many ways to launch a scan. What follows is a very quick example to illustrate how easy it is for Nessus to do a scan.

To verify that the Nessus client and Nessus daemon can talk, we will invoke the client to ask the daemon for a list of scanning sessions (of which there should not be any). If there is a communications problem, such as the wrong port, IP address, username or password, we will get an error. If things are configured correctly, we will then move on. Here is the command to try to log onto our Nessus daemon (on the same server) with an example username and password of user and pass:

```
nessus -m -q 127.0.0.1 1241 user pass
```

If the Nessus client and daemon are on the same server, this command can be modified using the -x option to disable certificate checks:

```
nessus -m -q -x 127.0.0.1 1241 user pass
```

Otherwise, Nessus may ask for guidance as to how it should attempt to encrypt the access between the Nessus client and daemon, with the following message:

```
Please choose your level of SSL paranoia (Hint: if you want to manage many
servers from your client, choose 2. Otherwise, choose 1, or 3, if you are
paranoid).
```

Choosing 2 allows one Nessus client to access multiple Nessus daemons as long as their username and password are known. Options 1 and 3 are for PKI-style deployments of Nessus daemons and are not covered in this article. The Nessus client will continue to prompt the user to verify whether a specific certificate is correct.

Normally, with a GUI tool, the GUI handles selection of which vulnerability checks to run and how the Nessus daemon should perform. This information is stored in a hidden configuration file named .nessusrc and is located in your home directory. Whenever a scan is run (from the GUI or from the command line), this file is overwritten. Without a GUI, though, invoking the Nessus client with the -m –q option is the right way to make the default configuration file for Nessus.

For a set of networks or IP ranges to be scanned, they must be specified and listed in one file. To specify an IP address, simply list it. To specify a range of addresses, CIDR blocks and ranges can be used. All target IP addresses need to be concatenated onto one line and should be saved into a file for use by the Nessus client. The following lines are example valid target IP specifications:

```
10.10.20.23
10.10.20.0/24
10.10.20.40-33
10.10.20-30.40-45
```

That last example would scan each class C network from 10.10.20.0/24 to 10.10.30.0/24, but would only scan hosts .40 through .45 within each network.

For our example, let's call the file to put this information into the "targets.txt" file. The Nessus client will need to know this, and which file it should output its data to.

```
nessus -q 127.0.0.1 1241 user pass ./targets.txt output.nsr
```

This will launch a scan against the systems contained on the first line of the targets.txt file. Watching the processes for nessusd while scanning is ongoing reveals what the particular instance of nessusd is attempting to do. For example:

```
nessusd: testing 127.0.0.1 (/usr/local/lib/nessus/plugins/roads_cgi.nasl)
```

could be a typical test.

While the Nessus daemon is active, a trace of its progress can be found by watching its logfile:

```
tail -f /usr/local/var/nessus/logs/nessusd.messages
```

Once the scan is completed, the Nessus client will write a report as shown in the example below:

```
127.0.0.1|http (80/tcp)|11408|INFO|;The remote host appears to be running a
version of;Apache 2.x which is older than 2.0.43;;This version allows an
attacker to view the source code;of CGI scripts via a POST request made to a
directory;with both WebDAV and CGI enabled.;;*** Note that Nessus solely
relied on the version number;*** of the remote server to issue this warning.
This might;*** be a false positive;;Solution : Upgrade to version 2.0.43;See
also : http://www.apache.org/dist/httpd/CHANGES_2.0;Risk factor :
Medium;CVE : CAN-2002-1156;BID : 6065;
```

The Nessus client defaults to writing an ASCII-style report with a relatively straightforward documentation of the vulnerabilities and hosts discovered. Other report formats, including HTML, a one-line style, and even XML, are also available.

Reports can also be converted on the fly as needed using the Nessus client. For example, to convert the output.nsr file from our example to an HTML file, the following command would be used:

```
nessus –i output.nsr –o output.html
```

In the example above, the output.html file could be served by any Web server, or loaded into any Web browser.

Using the X Windows GUI

The Nessus client, if compiled with the GTK toolkit, also provides a GUI for scan configuration and analysis of the discovered vulnerabilities. The GUI allows the Nessus user to select which Nessus daemon will be used for testing, which vulnerabilities will be used during the test, and how the Nessus daemon will perform the vulnerability scanning process. Many of these options are discussed in the next section.

The Nessus GUI can also be used to evaluate the results of a vulnerability scan. Information discovered during a vulnerability scan can be sorted by network addresses, DNS domain names, IP addresses, open ports, and discovered vulnerability types and severities. The GUI can also be used to save and convert Nessus reports in a variety of formats.

Tweaking Nessus: The Nessusrc File

Nessus's performance, accuracy, and intrusiveness can all be controlled from the nessusrc file. When the Nessus GUI client runs, it creates the nessusrc file with the options selected by the Nessus user. If no options are specified by the Nessus client, the Nessus daemon will choose a variety of its own default values.

To build customer nessusrc files, an easy technique is to first run a basic scan with the Nessus GUI client, and then to use the generated nessusrc file as a template that can be modified. The default nessusrc file will be located in the user's home directory as a hidden file named .nessusrc. For example, a root user would find their nessusrc file in /root/.nessusrc.

The nessusrc file has several sections. These are SERVER_PREFS, SCANNER_SET, PLUGINS_PREFS, and PLUGIN_SET. The SERVER_PREFS controls a variety of settings that relate to how the Nessus daemon performs scanning. The SCANNER_SET and PLUGIN_SET sections identify which NASL scripts are enabled and disabled. The PLUGIN_PREFS section provides information, such as a desired SNMP community string, which may be used by one or more NASL scripts.

Several keywords in the SERVER_PREFS section are useful in configuring a Nessus scan. Here is an example SERVER_PREFS section from a nessusrc file:

```
begin(SERVER_PREFS)
port_range = 1-1024
optimize_test = yes
safe_checks = yes
max_hosts = 30
max_checks = 10
cgi_path = /cgi-bin:/scripts
report_killed_plugins = yes
language = english
per_user_base = /usr/local/var/nessus/users
checks_read_timeout = 15
delay_between_tests = 1
non_simult_ports = 139
plugins_timeout = 320
auto_enable_dependencies = yes
use_mac_addr = no
save_knowledge_base = no
kb_restore = no
only_test_hosts_whose_kb_we_dont_have = no
only_test_hosts_whose_kb_we_have = no
kb_dont_replay_scanners = no
kb_dont_replay_info_gathering = no
kb_dont_replay_attacks = no
kb_dont_replay_denials = no
kb_max_age = 864000
plugin_upload = no
plugin_upload_suffixes = .nasl
save_session = no
save_empty_sessions = no
host_expansion = ip
reverse_lookup = no
detached_scan = no
```

```
continuous_scan = no
unscanned_closed = no
diff_scan = no
log_whole_attack = no
slice_network_addresses = no
end(SERVER_PREFS)
```

To limit or increase Nessus's activity (and required resources), the keywords max_checks and max_hosts can be modified. Increasing max_checks will allow the Nessus daemon to launch more processes per tested system, and increasing the max_hosts variable will increase the total number of systems a Nessus daemon will check at once. On large servers with extra memory, bandwidth, and CPU resources, a Nessus scan may run much faster with higher values than the defaults.

There is no exact formula for tuning a Nessus scan for optimized performance. However, during a scan, if CPU utilization is low, increasing these numbers by 50% may decrease the amount of time to complete a scan. Also, if the desired scan is not a full scan, but just a quick check for a few NASL scripts, increasing the max_hosts value may decrease the overall scan time.

The optimize_test keyword invokes logic at the Nessus daemon such that it will only attempt to launch a vulnerability check if a dependency has already been discovered. For example, with this option enabled, a "writeable anonymous FTP directory" check will not get invoked unless the "anonymous FTP access" check has returned positive. Similarly, the auto_enable_dependencies keyword can be used to enable a plugin that is required during testing. In the above example, if a Nessus user attempted to do a "writeable anonymous FTP directory" sweep of a network but forgot to enable the "anonymous FTP access" check, the scan would not return meaningful results. This keyword allows Nessus users to perform quick checks without having to memorize the potentially intricate dependencies some Nessus NASL plug-ins have.

The safe_checks keyword allows the Nessus daemon to select code within some NASL scripts that is less intrusive but possibly more prone to false positives. For example, a particular NASL script to look for a buffer overflow could attempt to actually flood an application with potential exploit data, or with safe_checks enabled it might read the banner information and make a decision about the vulnerability based on that.

If the particular scan includes a port scan, the ports to be probed are indicated with the port_range keyword. Ports can be listed individually or in ranges separated by commas. For example, 22,25,80,443,31330–31430 would scan the default ports for SSH, SMTP, HTTP, HTTPS, and all ports between and including 31330 and 31430.

In addition, the unscanned_closed option causes the Nessus daemon to treat ports that were not scanned as if they were closed. This is advantageous for doing "point" scans. For example, when using a Nessus scan to find SMTP servers on port 25, NASL scripts that perform checks on ports other than 25 would automatically not be invoked.

The cgi_path keyword can be used to optimize certain NASL scripts that attempt to find vulnerabilities on Web servers with CGI-BIN paths in non-obvious places. This variable can be modified if the Nessus user has knowledge about how a particular Web server has configured its CGI-BIN location.

The last keyword in the SERVER_PREFS section is the slice_network_addresses command. When enabled, the order in which IP addresses are tested is randomized. In an

environment which is not reactive to a Nessus scan, this has no effect on results of the scan. However, some dual-homed servers may experience several full scans at the same time and suffer some sort of negative impact. Consider a router, switch, or even a database server which is multi-homed on more than one network. If such a system were hit on each interface at the same time, it might experience slower network response time, slower server response times, or possibly even some sort of denial of service.

The PLUGIN_PREFS section specifies a wide variety of options for the enabled Nessus plug-ins. Some of these can be used to increase the accuracy and number of vulnerabilities discovered on any network. For example, Nessus can also be preconfigured with additional information, such as an SNMP community string and Windows network "auditing" account information. This information can be used by the Nessus daemon to actually log on to the tested hosts and check specific settings.

In the case of SNMP, several dozen Nessus checks make direct queries to the SNMP information for security information. To specify an SNMP community string to be used during testing, the SNMP port scan[entry]:Community name : = entry would be coded with the desired string. It is important to realize that there still may be other NASL scripts which do SNMP checks that do not take advantage of this setting, but that there are one or more checks that do. In the specific case of SNMP, there is a specific NASL plugin that attempts to brute-force several common SNMP community names. However, there are several dozen other NASL SNMP checks that use any discovered SNMP community names, and any preconfigured in the nessusrc file.

For Windows auditing, if a Primary Domain Controller is being used, then a global account which has read-only registry rights to each server in the domain can be used by Nessus to actually log in to an evaluated system. This account's username, password, and domain are placed in the nessusrc Login configurations options, specifically, within the PLUGIN_PREFS section. These entries are shown below:

```
Login configurations[entry]:SMB account : =
Login configurations[password]:SMB password : =
Login configurations[entry]:SMB domain (optional) : =
```

When Nessus sees that a target host is a potential Windows host, it will attempt to log in to the system and make registry calls to determine the configuration and whether any vulnerabilities exist.

(For more information, see "Utilizing Domain Credentials to Enhance Nessus Scans," by Ty Gast of the Security Assurance Group, located at *http://www.nessus.org/doc/nessus_domain_whitepaper.pdf)*.

To determine whether a host is alive, the following settings can be configured for Nessus:

```
Ping the remote host[entry]:TCP ping destination port(s) : = built-in
Ping the remote host[checkbox]:Do a TCP ping = yes
Ping the remote host[checkbox]:Do an ICMP ping = no
```

If either a TCP ping or an ICMP ping check is enabled, then one of those checks has to return positive, or else the tested IP address will be assumed not to be active. This is important when testing systems that are protected by firewalls.

From outside a firewall, or with the presence of host-based firewalls, this may be the only way to accurately test a host's vulnerabilities. Consider a host serving a SQL data-

base on a high port that also has a firewall in front of it which prevents any access to the server except via SQL. Any attempt to ping it with an ICMP packet, make a TCP connection on any port other than the SQL port, or send any UDP packet to it would not be successful. However, if SQL were running on the default SQL port, the first NASL plugin that connected to it would succeed and the vulnerabilities would be tested.

When scanning a large network, choosing either ICMP or TCP host enumeration is recommended. Otherwise, each NASL plugin script must time out on its own. This makes for a much longer, but also much more accurate, Nessus scan. For a TCP ping, Nessus will attempt to connect to several common ports in the 1–1024 range. If desired, a customer port list can be placed in the TCP ping destination port(s) configuration option.

There are several hundred more settings for the PLUGIN_PREFS section. Readers interested in crafting custom settings for other variables are encouraged to experiment with the Nessus GUI client and observe the changes made in the produced nessusrc files. Readers are also encouraged to consult the Nessus mailing list archive highlighted at the end of this article.

The last two sections in the nessusrc file are PLUGIN_SET and SCANNER_SET. Both of these sections list which NASL plug-ins are enabled and disabled. The SCANNER_SET section is used to identify plug-ins that perform some sort of port scanning. These are executed before the regular vulnerability-checking plug-ins. Historically, Nessus has been able to invoke a wide variety of port scanning and host enumeration tools, such as NMAP (*http://www.nmap.org*) and SNMPWALK. Currently, Nessus 2.0 does not require any external port scanning tools. When listing a plugin for testing, the ID will be printed followed by a simple "yes" if it is to be considered for testing, or a "no" if it is to be excluded. Here is an example excerpt from a nessusrc file:

```
10909 = no
10330 = no
11268 = no
11480 = yes
10351 = yes
10010 = yes
10536 = yes
10440 = yes
11210 = yes
11641 = yes
11554 = yes
```

Nessus Vulnerability Checks and NASL

There are almost 2000 unique Nessus NASL scripts available. These scripts are each labeled with a unique Nessus ID, contain a brief description of the relevant security audit or vulnerability, and also have links to CVE (*http://cve.mitre.org*) and Symantec's BugTraq (*http://www.securityfocus.com*). Each NASL script is grouped into a family that focuses on specific services, operating systems, and "problem areas" such as peer-to-peer (P2P) file sharing.

Many of the vulnerability checks performed by NASL are unique tools unto themselves. For example, the traceroute tool includes logic to perform multiple traceroute checks to many hosts without colliding. There are checks that identify wireless access

There are almost 2000 unique Nessus NASL scripts available.

points by their TCP/IP operating-system fingerprint, individual Web interface, and SNMP or FTP banner information.

Nessus includes plug-ins that can perform extremely complex audits of Web services. For example, the "torture" CGI NASL script actually reads in the values of the Web forms automatically found during a Nessus scan, and commonly queries these forms with typical CGI-BIN exploit techniques. Similar checks are performed by commercial Web-application testing tools.

Deploying Permanent Nessus Daemons

When deploying Nessus, many organizations take advantage of placing a server permanently within an infrastructure that is dedicated to scanning. These servers are often located within server farms, behind firewalls, and within DMZs. Sometimes they are also deployed permanently outside of a network to perform an external audit.

The advantage to deploying dedicated servers is that they are always on and ready to perform a scan from the perfect location. Anyone who has performed a network security audit and has had to drag a laptop around a network to get to the right spot to perform a scan knows what a hassle it can be. By pre-placing a Nessus within a network, the time it takes to perform a scan is just the time it takes to perform a scan. There is no more need to walk to the building that has the firewall in it, fly to the remote location, or find the proper network cable or switch port to plug into.

The systems that run the Nessus daemons should be secured so that no remote services are running except those that are required to securely communicate with the device. For example, a common "hardened" Nessus daemon server would include the actual "nessusd" process, which normally listens on port 1241, and the Secure Shell daemon, which listens on port 22. It is also common for users to use a system firewall to limit connections to the Nessus daemon from the authorized network users and to limit access to port 22 from only "trusted" IP addresses within the network.

When a Nessus daemon is deployed, very little vulnerability information will be present on the device. Most scan data passes through the Nessus daemon and is not stored there.

Minimizing Scanning Impact

Scanning with any network vulnerability scanner, including Nessus, can have a negative impact on the target network. This impact can come from an overall slowdown or degradation in network performance, or a denial-of-service event on one or more devices.

When Nessus 1.0 was originally available, its standard tests included a list of DoS tests that were enabled by default. Often, a Nessus user in the late '90s would end up finding out very quickly if their network systems were vulnerable to the "ping of death" DoS attacks and several others. With Nessus 2.0, great care has been taken to limit the number of potentially damaging DoS checks that Nessus can perform. DoS checks are no longer enabled by default.

When Nessus performs its checks, the traffic that it generates may have a negative impact on a switch or firewall. Not only does the scan send packets across a network, but often, one packet in a port scan will be considered a new session by a switch or firewall. For example, some appliance firewalls that keep track of unique network sessions will quickly lose track of their current valid sessions when faced with a large port

scan. If a firewall can only keep track of 65,000 unique network sessions, a port scan of one host can quickly fill this buffer up, pushing out valid sessions. Similarly, poorly designed firmware in routers, switches, and some firewalls doesn't handle the large numbers of network sessions generated during a vulnerability scan. The flaws in the handling can be very subtle.

For example, the author used to work for a certain routing vendor, and one of our products would reboot when scanned with a certain commercial vulnerability scanner. Duplicating the same scan with Nessus or NMAP did not cause the reboot, and only doing a low-level packet trace resulted in identifying the exact sequence of events which brought the network device down.

To avoid stressing the network infrastructure, Nessus daemons should be placed as close as possible to their desired targets. Any deployment that minimizes the flow of host identification, port scans, and vulnerability checks across the core routers and switches should be considered. Also, when doing a vulnerability scan for the first few times on a large enterprise network, attempt to correlate the scan with the CPU load, packet rates, and any other statistics that can be obtained from network engineering. These statistics may indicate faults in the network before actual faults are caused, if any.

Finally, when a specific host is identified, conducting vulnerability scans on the host may cause some of its applications or the entire operating system to crash. Even with careful writing and testing of NASL scripts, and with careful laboratory testing, there is no way to guarantee that an NASL script will have no impact on every possible system and application. All vulnerability scanners have this problem. Even if there were a way to perform regression testing on all known operating systems and applications, there are still thousands of custom applications that the Nessus project does not have access to test.

When testing a production network, a lab test on a certain group of NASL scripts can be used as a benchmark to predict the impact of running the NASL scripts on the entire network. If a network has many similar hosts, such as a large network of Red Hat Apache servers, doing some quick tests in a lab before blasting the entire network can give a sense of the impact that will result. However, in networks with extremely diverse families of servers, it is impossible to predict the impact of a scan without actually doing the scan itself.

Unlike other vulnerability scanners, Nessus does attempt to alert the user if it may have crashed a server during testing. It does this by looking for changes in the available ports of a tested server. If during a port scan, it sees that ports 80, 443, and 6000 are open, it will alert the user that the server (or service) may have crashed if one of the ports goes away.

Readers are advised that a poorly run network may crash for many reasons, including failing to survive a vulnerability scan. They should not be dissuaded from performing active scans, but should realize that doing any network activity can impact the network. Most vulnerability scans simply exercise systems that have not been used in a long while, or "operational" systems that were not fully tested.

The Future of Nessus

The Nessus project is currently funded by Tenable Network Security (*http://www.tenablesecurity.com*), which is a commercial network security product company.

Although the Nessus project continues to be open sourced, Tenable has introduced several commercial products that greatly enhance Nessus for large enterprises.

Long-term plans for Nessus include a preliminary design of Nessus 3.0, which will continue to make Nessus one of the top vulnerability scanners available to the network security community. In the short term, the Nessus Web site recently added the ability for users to submit information privately and anonymously about which of their NASL scripts have produced false positives for them. This helps Tenable and the Nessus project quickly identify when a NASL script needs to be modified. Nessus 2.0 was released early in 2003, and ports to the OS X operating system have been completed.

There is a wide variety of information about Nessus available at *http://www.nessus.org*, as well as a searchable mailing list located at *http://msgs.securepoint.com/nessus/*.