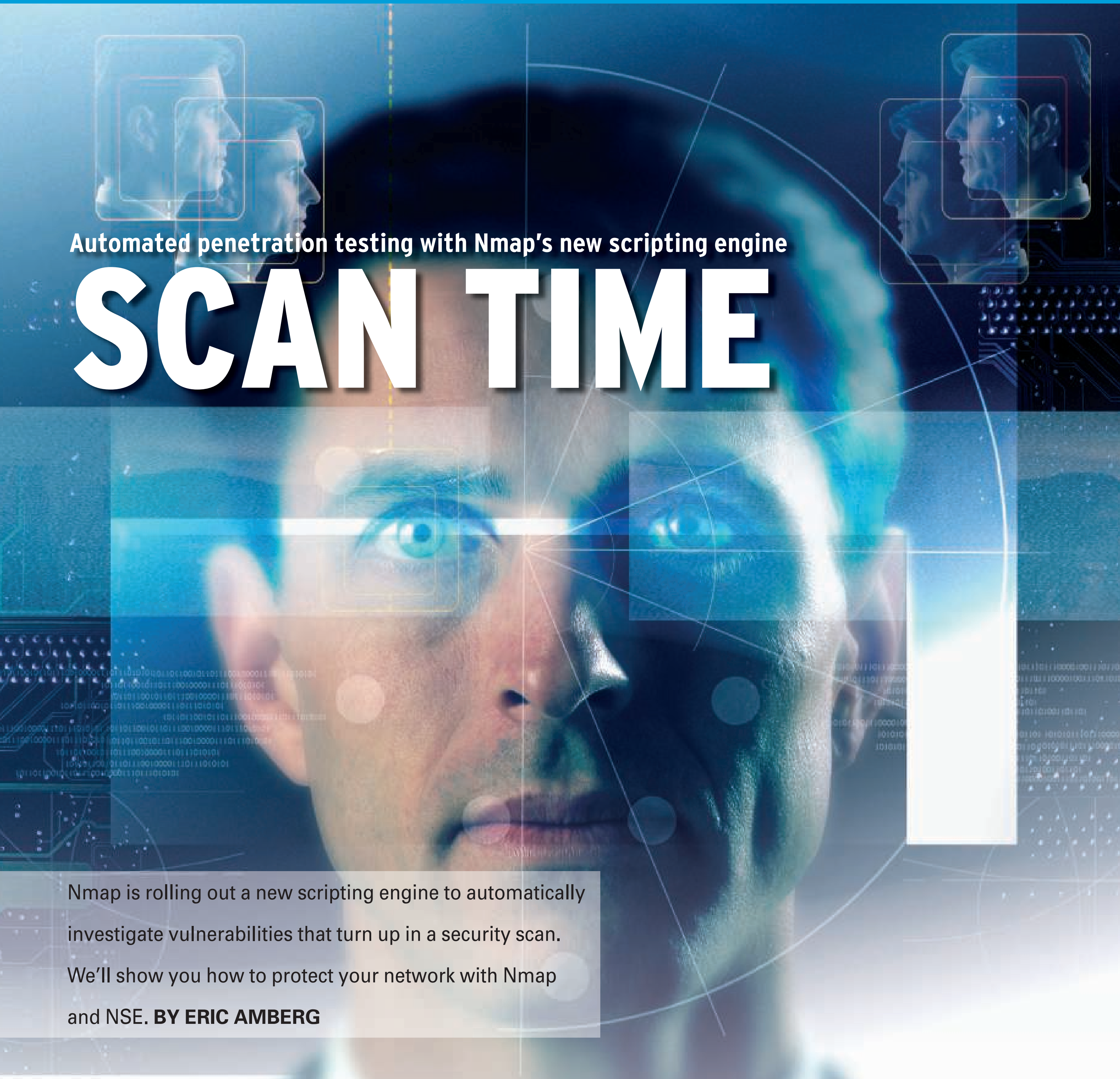


Automated penetration testing with Nmap's new scripting engine

SCAN TIME

Nmap is rolling out a new scripting engine to automatically investigate vulnerabilities that turn up in a security scan. We'll show you how to protect your network with Nmap and NSE. **BY ERIC AMBERG**

Nmap is the tool of choice for penetration testing [1]. Experts use Nmap to search out security holes and scan for open network services. But what happens when you find a problem? Many administrators prefer to follow up the discovery with additional tests. For instance, if Nmap finds an http service, why not query to determine the web server version?

Thus far, administrators have written their own scripts to parse Nmap output files – a slow and time-consuming process. The Nmap project recently decided it was time to introduce a scripting engine so that users could automate Nmap functions into custom scripts. Fyodor, the Nmap project leader, placed the development of this scripting engine in the capable hands of Diman Todorov. The result is the Nmap Scripting Engine (NSE) [2], which has been an integral part of Nmap since version 4.21.

NSE extends the core functionality of the Nmap scanner, providing detailed information on services such as NFS, SMB, or RPC. You can also use NSE to search for active systems using domain lookups, Whois searches, or other source network discovery techniques. To discover backdoors, NSE checks any version strings it detects against regular expressions – a useful option for pen testers who want to check existing vulnerabilities by launching exploits. However, the developers point out that Nmap is not looking to compete with the Metasploit framework [3].

NSE provides an easy means for building automated solutions around Nmap. The NSE option works well for small to mid-sized networks. Tools such as Nessus [4], GFI LANguard [5], or ISS Internet Scanner [6] might be better suited for large-scale scanning operations. You can download the NSE source code from the project server [1] or check out your distributor's repository for a binary.

NSE is built on the Lua interpreted scripting language [7]. Lua is designed to work with software written in other languages, such as C or C++. Lua, which is based on ANSI C, owes its flexibility to the fact that most functions are available in the form of libraries. This approach makes it easy to extend the functional scope by binding required modules. In addition, the Lua interpreter has a very small footprint; even commercial games (including World of Warcraft) rely on Lua.

A Lua interpreter forms the core of NSE (see Figure 1). When Nmap detects a host or port, it calls the Lua interpreter with a matching script to leverage the abilities of the Lua language and Nmap-specific functions from the NSE library. The NSE library provides additional Nmap-related features that Lua doesn't offer such as tools for evaluating and manipulating IP addresses, using Perl-compatible regular expressions, or manipulating URLs.

Lua makes intensive use of data structures in tables. Tables contain attribute-value pairs, and they can also contain subtables. For example, NSE uses the *host* and *port* tables to access Nmap scan results. A *registry* table that all scripts can read and write to resides above all scripts. This design lets scripts exchange data.

NSE in Action

Nmap and NSE use the Nmap API to exchange information, including the target host name, the operating system and IP address, the port number, and the port status. The API also lets users call Nmap's socket functions for network communications. An extension that lets users send custom packets is due for release in the near future.

Nmap starts by checking whether it can reach ports on a host. In case of a TCP scan, the tool will ascertain the port status, which could be *open*, *closed*, or *filtered*. Once Nmap has detected a port, and assuming NSE has been enabled with the *-sC* option, the subsystem will attempt to locate a matching script for the test.

Scripting rules determine whether Nmap will run an NSE script. These scripting rules specify the conditions under which the script will launch.

You can build your own scripts or rely on the pre-built scripts included with NSE. The pre-built scripts attend to common tasks often associated with Nmap. For example, if the scanner discovers that TCP port 80 is open and if the pre-built *showHTMLTitle.nse* script is available, a call to the script is issued. The script queries the web server's front page and displays the header (see Figure 2). Several pre-built scripts are available in */usr/share/nmap/scripts*.

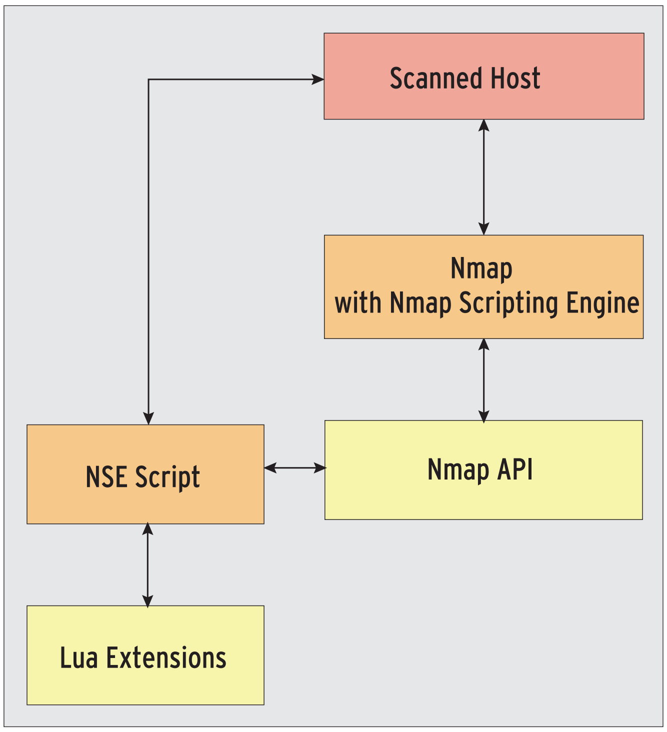


Figure 1: A combination of NSE scripts, Nmap libraries, and new Lua calls extends the functional scope of the classic Nmap tool.

Categories

NSE organize scripts in categories to allow for more granular control. The defined script categories are *safe* and *intrusive*, *malware*, *version*, *discovery*, and *vulnerability*. Scripts that NSE categorizes as *safe* are very unlikely to cause problems on the scan target. The scripts in the *intrusive* category are unlikely to cause damage, but will attempt to use default passwords to access systems, and thus they are more likely to generate log entries on the scan target.

Scripts in the *malware* and *vulnerability* categories test the scan targets for malware and known vulnerabilities. The *version* category plays a special role: scripts in this category, which are enabled by the *-sV* command-line option, extend Nmap's version detection capabilities. Their output is no different from standard Nmap output; the script is not referenced in the report generated by the scan. The *discovery* category includes scripts that are designed to find out more about the target host or network by querying various services (including SNMP or LDAP).

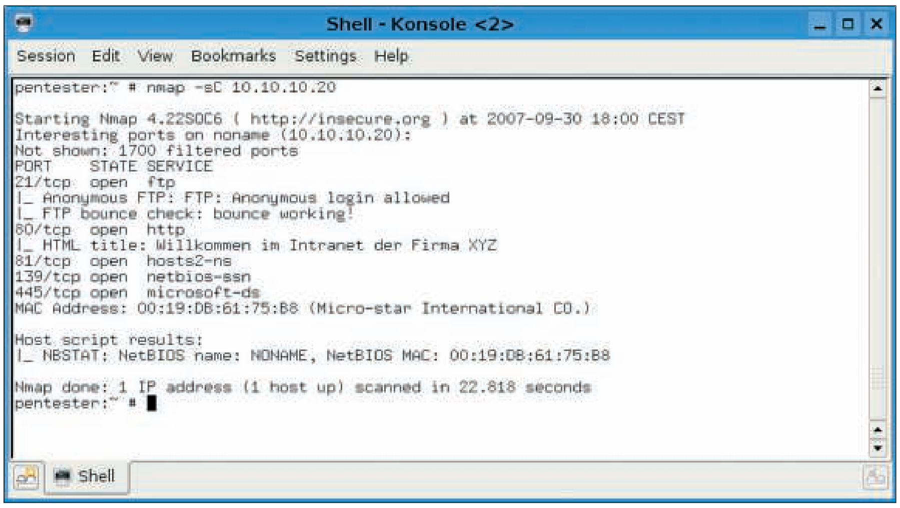
The *--script* argument specifies the categories, directories, and individual scripts for NSE to integrate and execute. A comma-separated list of values is possible. Nmap starts by searching for a category with the specified name; if the search fails, Nmap then looks for a directory with the same name and binds any scripts ending in *.nse*.

If this search also fails, Nmap searches for an individual script of the same name. For example, the command-line option *--script discovery,malware* binds the categories *discovery* and *malware*. A database file titled *script.db* in the *scripts* subdirectory maps scripts to individual categories. Admins can update the file by entering *--script-updatedb* after adding scripts. If you set the *-sC* flag, only scripts from the categories *safe* and *intrusive* are executed.

Your Own NSE Scripts

Anexample will make it easier to understand the structure of an NSE script. Most built-in scripts are kept simple – normally smaller than 100 lines including comments. The *ripeQuery.nse* script in Listing 1 should give you some idea of how Lua and NSE work. The script queries an address for an entry in the RIPE network registration system [8].

An NSE script consists of three components: the *header* defines the script name in the Nmap output, the script category, and the run level. The *rule* specifies the conditions under which the script is executed. The *action* component calls the functions that handle the actual



```
pentester:~ # nmap -sC 10.10.10.20

Starting Nmap 4.22SOC6 ( http://insecure.org ) at 2007-09-30 18:00 CEST
Interesting ports on noname (10.10.10.20):
Not shown: 1700 filtered ports
PORT    STATE SERVICE
21/tcp  open  ftp
|_ Anonymous FTP: FTP: Anonymous login allowed
|_ FTP bounce check: bounce working!
80/tcp  open  http
|_ HTML title: Willkommen im Intranet der Firma XYZ
81/tcp  open  hosts2-ns
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 00:19:DB:61:75:B8 (Micro-star International CO.)

Host script results:
|_ NBSTAT: NetBIOS name: NONAME, NetBIOS MAC: 00:19:DB:61:75:B8

Nmap done: 1 IP address (1 host up) scanned in 22.818 seconds
pentester:~ #
```

Figure 2: In new script mode (option -sC), Nmap calls external scripts. Nmap has detected that port 80 is open, so the tool chips in to query the heading of the website it has discovered and displays the results. The figure also shows NSE detecting the NetBIOS names of a target.

tasks. Authors can implement all three sections of the script in Lua.

Each NSE script has a header that comprises four descriptive details. The *id* contains the script name and is used for Nmap output; *description* is a short description; and *author* names the author. The interpreter does not process the *license* field.

A script will only run under specific conditions. Port and host rules control its behavior. The example in Listing 1 defines a function that expects the Lua *host* and *port* tables in lines 11 through 13. The tables contain the values discovered by Nmap in the course of a scan. For example, *host.ip* supplies IP addresses, and *port.number* the port for the current scan.

The *hostrule* function thus calls the library function *ipOpts.isPrivate* to check to see whether the current IP target is a private address in the sense of RFC 1918. (A RIPE test will not make sense if the address is private.) NSE then runs the body of the script, the *action* function that starts in line 15.

If you are not interested in investigating the host but just need details of a single service, it makes sense to use a line such as:

```
portrule = shortport.➥
port_or_service(21, "ftp")
```

To do so, enable the Lua *shortport* extension by calling *require "shortport"*.

The beef of the script follows the header details for the *action = function (host, port)* function. The function again expects *host* and *port* as arguments. In line 16, the script first creates a new TCP/IP socket then defines the local variables *status*, *line*, and *result*. Line 20,

```
socket:connect➥
("whois.ripe.net", 43)
```

tells the script to connect to the RIPE database, which responds to TCP port 43 requests. The Whois service expects an IP address in quatted dot notation. The code *socket:send(host.ip .. "\n")* sends the IP address. The query line must be terminated by a newline character. To allow this to happen, the concatenator operator .. adds *\n* to the IP address for the scanned host parsed from *host.ip*. Inside an infinite loop beginning in line 23,

```
local status, lines = ➥
socket:receive_lines(1)
```

reads the response from the RIPE database until the data input dries up. The Boolean variable *status* shows whether the connection is still delivering data.

If this is not so, *break* in line 27 quits the loop. The parsed response is available in *lines*, and *result* accumulates the

Listing 1: ripeQuery.nse

```
01 require “ipOps”
02
03 id = “RIPE query”
04 description = “Connects to the
   RIPE database,
05 extracts and prints the role-
   entry for the IP.”
06 author = “Diman Torodov
   <diman.todorov@gmail.com>”
07 license = “See nmaps COPYING
   for licence”
08
09 categories = {“discovery”}
10
11 hostrule = function(host, port)
12 return not ipOps.
   isPrivate(host.ip)
13 end
14
15 action = function(host, port)
16 local socket = nmap.new_socket()
17 local status, line
18 local result = “”
19
20 socket:connect(“whois.ripe.
   net”, 43)
21 socket:send(host.ip .. “\n”)
22
23 while true do
24 local status, lines = socket:
   receive_lines(1)
25
26 if not status then
27 break
28 else
29 result = result .. lines
30 end
31 end
32 socket:close()
33
34 local value = string.
   match(result, “role:(.-)\n”)
35
36 if (value == “see http://www.
   iana.org.”) then
37 value = nil
38 end
39
40 if (value == nil) then
41 value = “”
42 end
43
44 return “IP belongs to: “ .. value
45 end
```

input. After all response lines are read, *socket:close()* closes the socket.

The next few lines are designed to discover what the entry for the submitted IP address says. The Whois server returns a large number of response lines; our example simply searches for lines that start with *role:*. To do this, the script calls the Lua *string.match()* function. The line

```
local value = string.match⮐
(result, "role:(.-)\n")
```

searches in the *result* variable for a line that matches the search key. The key starts with *role:*; however, the function only returns the remainder of the line in parentheses. To keep this example simple, I edited a script that is provided with the Nmap distribution. To be more robust, scripts should always perform a plausibility check on returned values.

Nmap API Treasures

The Nmap API lets NSE scripts communicate with Nmap. The Lua *host* and *port* tables provide access to critical values such as the IP address, port, service, or port status. The Lua *host* table returns the values shown in Table 1, assuming that Nmap has collected some values; otherwise, it returns an empty string. This also applies to details of active ports, which are stored in the Lua *port* table, along with attributes for the protocol, service, and status.

Table 1: Host and Port Table Attributes

Attribute	Type	Meaning
Host Table Attributes		
host.os	String	Contains a string with the detected operating system name if Nmap is launched with the *-O* option
host.ip	String	Contains the IP address for the current scan target host
host.name	String	Contains the host names returned by a reverse lookup
host.mac_addr	String	Contains the MAC address for the target host
Port Table Attributes		
port.number	Integer	The port currently being scanned
port.protocol	String	Either *tcp* or *udp*
port.service	String	Contains a service name if Nmap was able to map a name in the scope of version detection
port.state	String	The port status can be *open* or *open|filtered*; the script is not run against closed ports and filtered ports

The sample shows how to call these values and integrate them with an NSE script. To communicate with other systems, NSE scripts need an interface. An API makes it possible to use the Nsock library (also used internally by Nmap).

Socket Programming

Legacy communications typically follow the socket approach. The following code

THE MATHEMATICS OF HUMOUR

TWELVE Quirky Humans,
TWO Lovecraftian Horrors,
ONE Acerbic A.I.,
ONE Fluffy Ball of Innocence and
TEN Years of Archives
EQUALS
ONE Daily Cartoon that Covers the
Geek Gestalt from zero to infinity!

Over Two Million Geeks around the world can't be wrong!
COME JOIN THE INSANITY!

UserFriendly.Org

creates a new socket dubbed *sock* and then binds the socket to the host using the specified port.

The protocol is optional and can be *tcp*, *udp*, or *ssl*:

```
sock = nmap.new_socket()
sock:connect(Host, Port ➘
[, Protocol])
```

The *sock:send(request)* code can then send a prepared request to the target host. The semantics of the line,

```
status, value = ➘
sock:receive_lines(lines)
```

which reads data from the socket, is fairly unintuitive.

If the requested lines, or a lesser number of newline-terminated lines, arrive before the timeout, they are stored in the variable *value*. However, if the network buffer has more lines than anticipated, the API will send more lines, as described in the documentation [2].

If successful, *status* contains a value of *true*. Finally, *sock:close()* closes the socket. The Nmap API has far more functionality, but most typical cases are covered by the examples provided.

Exception Handling

The Nmap API also implements an exception handling mechanism for catching errors and canceling scripts that would provide erroneous results because of incorrect conditions.

Exceptions are important for robust scripts, in that services that you scan across a network will not always respond as expected. A glance at the *finger.nse* script shown in Listing 2 illustrates how exception handling works.

If an error occurs on the target system or the connection times out, the exception handler closes the socket gracefully.

Lua Extensions

Whereas the Nmap API handles communications between Nmap and the NSE scripts, the Lua extensions add significant functionality to NSE. The modules are located below the *nselib* subdirectory. An NSE script binds the modules via *require Modulename*. Table 2 lists the available extensions. It is not important whether you bind modules before or after the header.

Table 2: Lua Extensions for NSE

Extension	Function
bit	Supports bit operations; for example, *bit.lshift(a, b)* performs a bitwise left shift by *b* digits for *a*.
pcre	Refers to Perl-Compatible Regular Expressions and supports their use. This lets scripts extract search keys from service responses.
ipOps	Supports operations related to IP addresses. As shown in the sample script, *ipOps.isPrivate(address)* checks to see whether the specified address is a private address in the sense of RFC 1918.
shortport	Provides standard tests for port rules. Many scripts use this option and run *shortport.portnumber(port)* to check the port.
listop	Lists processing as used by other programming languages such as Lisp or Haskell. The developers refer to this extension as experimental.
strbuf	Supports certain string operations; for example, *strbuf.eqbuf(string1, string2)* compares two strings.
url	Extends Lua's URL manipulation abilities, adding functions that create or analyze parameter lists.

The script starts by calling a function whose only purpose in life is to close a socket in case of error. The exception handler is created by the line:

```
try = nmap.new_try()
```

The preceding line expects an *err_catch()* function argument, which closes the socket. If a function returns an error, the interpreter cancels the function without a comment. The function passed in as an argument specifies the actions to perform in this case.

NSE provides a powerful and flexible extension that administrators can use to design custom Nmap scans. Modules from the NSE extensions library are perfect for extending the basic functionality of the underlying Lua script language. The Nmap API gives programmers the ability to reference the *host* and *port* tables and thus parse scan results. The API also adds numerous communications options for scripts.

Listing 2: finger.nse (Excerpt)

```
local err_catch = function()
socket:close()
end

local try = nmap.new_try(err_catch())

socket:set_timeout(5000)
try(socket:connect(host.ip,
port.number,
port.protocol))
try(socket:send("\n\r"))

status, results = socket:receive_lines(100)
socket:close()
if not(status) then
return results
end
```

Conclusions

NSE scripts are quickly conquering the former domains of security tools such as Nessus. A stable version of NSE is not available as of this writing.

Nmap 4.21 contained the alpha versions and 4.22 was part of Google's Summer of Code.

The stable Nmap 4.5 version will include NSE, and it is scheduled for completion by the time this issue of *Linux Magazine* hits the shelves. NSE definitely has the potential to become an indispensable part of Nmap. ■

INFO

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

[2] NSE documentation: *http://www.insecure.org/nmap/nse/*

[3] Metasploit Project website: *http://www.metasploit.org/*

[4] Nessus project: *http://www.nessus.org*

[5] GFI LANguard: *http://www.gfi.com/languard*

[6] ISS Internet Scanner: *http://www.iss.net/products/Internet_Scanner/product_main_page.html*

[7] Lua: *http://www.lua.org/*

[8] RIPE: *http://www.ripe.net/*