Writing Nasl Scripts

Introduction:

Nessus is one of the highly respected vulnerability scanners in the security world today. Looking at the history of Nessus, Nessus project was started way back in 1998 by Mr. Renaud Deraison. From 1998 till today, nessus has become a mature vulnerability scanner with some unique features. Nessus has won numerous awards, including the 2002 Network Computing "Well Connected" award and PC Magazine's 2003 "Open Source Product of the Year" award. This paper includes

- What is Nessus?
- What is Nasl?
- Why to write your own Nasl script?
- What are the applications you need to write nasl script
- Loading nasl script into nessus server
- Writing first nasl script
    - Configuration parameter used in nasl script
    - Post configuration – Including header files
    - Security check and how to report texts from script
- Few debugging tips
- Where to find a complete guide for nasl?

Let us first understand what is nessus?

What is Nessus?

If I was writing this paper before a year, I could have said it as “Nessus is an open source vulnerability scanner” but now it is not advisable to write open source as Nessus is now divided in two different versions. One version of Nessus is still open source but the other version of Nessus is close source. It is out of scope of this paper to cover why it was open source in the past and why it is close source now? I think there is enough discussion in the security arena on this. There are enough details on this subject on official nessus site as well.
Design of nessus is divided in to two parts.

- Engine
- Plugins (Nasl scripts)

Engine takes care of everything except security checks. Basically, engine reads each security check (known as plugin) from a file depending on the scan configuration and runs the check. Engine also takes care of updates and licensing. Plugins are security checks written in language supported by Nessus engine (Nasl). This paper covers the second part.

What is Nasl?

NASL stands for **N**essus **A**ttack **S**cripting **L**anguage. It is a scripting language supported by nessus which can be used for writing security checks. These security checks are divided into different groups known as plugin family.

Why to write nasl scripts?

Over the years, lot of plugins have been written for Nessus. Currently when user installs Nessus, it also installs all these plugins. People submit their plugins with nessus, so the user needs to download the new plugins regularly. Still there are cases where user wants to write his own security checks. In a nutshell one should write nasl to

- To contribute to the other people so that they do not need to invent the same wheel again
- It will help in understanding vulnerability in detail

Author: Hemil Shah hemilmshah – at – yahoo.com
Review: Niyati Dalal niyatidalal – at – hotmail.com

Application required for writing nasl script:

Writing nasl script is very easy once you are aware of the syntax. To write a nasl script you need

- An editor. vi editor on Linux or notepad on windows will do.
- Idea of security check

After writing a script, save the file with extension “nasl”. Most of syntaxes for nasl are like C language. Now let us see how to load script in to nessus server.

How to load nasl script in to nessus server:

- Write the script, save the file with .nasl extension.
- Copy this file to plug in directory where nessus server is installed. Default path for it on linux machine will be /usr/local/lib/nessus/plugin/
- Restart the nessus server
- Restart the nessus client

Writing first NASL script

Typical NASL script can be divided into three parts

- Configuration
- Post configuration – including header
- Security check

Before we dig more into each of the above mentioned parts, there are a couple of things one should keep in mind.

- All statements of nasl end with semicolon.
- To comment any line in NASL script, use “#” at the beginning of the line

Now let us understand each of above mentioned part in detail

Configuration:

First part of any NASL script is configuration.

To write a nasl script one needs to set its configuration first. This is the configuration which nessus server uses to run a nessus check. Configuration includes - plugin information (ID, Name and description), category, family, dependencies and prerequisites. Typical configuration will look like

```
If( description )
{
script_id(12248);
script_version ("$Revision: 1.5 $");
script_name(english:" notes.ini checker");
desc["english"] =
" This plugin attempts to determine the existence of a directory traversal bug on the remote Lotus Domino Web server.

Risk Factor : High”;

script_description(english:desc["english"]);
script_summary(english:" notes.ini checker ");
script_category(ACT_ATTACK);
script_family(english:"Misc");
script_copyright(english:"This script is Copyright (C) 2004 Net-Square Solutions Pvt Ltd.");
script_dependencie("http_version.nasl");
```

Author: Hemil Shah hemilmshah – at – yahoo.com
Review: Niyati Dalal niyatidalal – at – hotmail.com

```
script_require_ports("Services/www",                                    80);
exit(0);
}
```

Now, let us understand each of these parameters in detail

- Script_ID: It must be unique. It is recommended to have an ID between 1- 100000. For e.g. in above case the script_id is 12248..
- Script_version: This is a version of the script. As we are writing the first version of the script, keep it version 1.0. In above case, the script is modified and its current version is 1.5. i.e. *script_version ("$Revision: 1.5 $");*
- Script_name: This is the name of the script. This will be shown in the list of plugins when user will connect to the nessus server. In above case, *script_name(english:" notes.ini checker");*
- Desc: This is the detail description of the script. What the script will check should be mentioned here.

  desc["english"] = *"This plugin attempts to determine the existence of a directory traversal bug on the remote Lotus Domino Web server ";*
  - script_description(english:desc["english"]);
- script_summary: This is the summary (single lined) description of the script.
- script_category: This defines category of the script. Possible values can be *"ACT_INIT, ACT_SCANNER, ACT_SETTINGS, ACT_GATHER_INFO, ACT_ATTACK, ACT_MIXED_ATTACK, ACT_DESTRUCTIVE_ATTACK, ACT_DENIAL ACT_KILL_HOST"*
- script_family – As mentioned before, all plugins are divided in to different families. Possible values can be "*Backdoors, CGI abuses, CISCO, Denial of Service, Finger abuses, Firewalls, FTP, Gain a shell remotely, Gain root remotely, General, Misc, Netware, NIS, Ports scanners, Remote file access, RPC, Settings, SMTP problems, SNMP, Untested, Useless services, Windows, Windows : User management*"
- script_copyright: Company name or an individual's name who owns the script. In above case, *script_copyright(english:"This script is Copyright (C) 2004 Net-Square Solutions Pvt Ltd.");*
- script_bugtraq_id: If the security check has bug traq ID, mention it in "script_bugtraq_id" i.e. script_bugtraq_id(4261);
- script_cve_id: If the security check has CVE ID, mention it in "script_cve_id" i.e. script_cve_id("CAN-1999-0502");
- script_dependencie: if the script is dependent on any other nasl script, mention the name of all those nasl scripts under this parameter. Use comma (",") to specify more than one script. In above case, *script_dependencie("find_service.nasl")*;
- script_require_ports: if script runs on particular service and requires particular port to be opened, that can be mentioned in this parameter. In above case, *script_require_ports("Services/www", 80);*

Post configuration – including header

If you have written any C code, you might be aware that one needs to include the header files to compile the program. Similarly, we need to include header files in nasl as well. If you are not familiar with C programming, do not worry. Header files are the files where functions are declared. Header files in nasl are known as inc files.
Syntax for adding header file is

```
include("http_func.inc");
```

There are ten different header files which are listed below

- dump.inc – This file has functions related to show data on standard output.

- ftp_func.inc – FTP functions
- http_func.inc – HTTP functions
- http_keepalive.inc – Extension functions for HTTP 1.1
- misc_func.inc – Functions for parsing service header and banners
- nfs_func.inc – NFS functions
- smb_nt.inc – SMB functions
- smtp_func.inc – SMTP functions
- telnet.inc – Telnet functions
- uddi.inc – Functions for formatting UDDI XML queries

Security check

This part of the script contains the actual logic of the script. User can use number of pre defined functions from header files and write security check. At the end, one needs to report the result of the script which the engine will use in the final report generation. Following functions can be used to write output to report.

1) Security_hole: Can be used to report server flaw error.
2) Security_warning: Can be used to report minor flaw.
3) Security_note: Can be used to report misc. information

All these functions take two arguments, first argument is port number and second argument is string, which is description to be shown in report.

Debugging nasl script:

After writing nasl script to check the syntaxes, use –t option of the nessus binary. Syntax for the same is

```
# nessus –t <Destination IP> /usr/local/lib/nessus/plugins/<scriptname.nasl>
```

This will check the syntax error and show the error if any along with the line number.

Where to find a complete guide for NASL

Earlier, complete reference guide was available from the official site nessus.org but now it is not available from nessus site and if it is available, at least I am not able to find it. The reference material was available in two formats – pdf and text. Searching Google for “nasl2_reference.pdf” will help you in finding NASL syntaxes.

Reference:

- Nasl2_reference.pdf

Author: Hemil Shah hemilmshah – at – yahoo.com
Review: Niyati Dalal niyatidalal – at – hotmail.com