Advanced Scripting Language

Un supporto per il Penetration Testing

(a cura di **Stefano Maccaglia**)

Cos'è il Penetration Testing?

- Secondo una definizione largamente condivisa il Penetration Testing è un metodo di valutazione della Sicurezza di un Sistema o di un'ambiente basato sulla simulazione di attacchi compiuta con gli stessi mezzi e con le stesse modalità usate dagli attaccanti reali.

- Le problematiche di Sicurezza identificate attraverso il Penetration Test potranno essere valutate dal punto di vista dell'esposizione, dell'impatto che hanno o potrebbero avere sulla struttura, nonché delle possibili soluzioni tese a risolverle o quantomeno a mitigarne gli effetti.

Modalità di Test?

- **White Box** – Il tester ha una approfondita conoscenza del network sottoposto ad analisi. Questa conoscenza pone in essere la peggiore situazione operativa, quella in cui l'attaccante conosce bene il "territorio". Questa condizione non è frequente, ma è sicuramente la più pericolosa.

- **Black box** – Il Tester non ha, in questo caso, nessuna conoscenza dell'ambiente nel quale verranno eseguiti i test.

- **Gray box** - Il Gray Box Testing simula il comportamento di un impiegato interno ed è utile per valutare i rischi provenienti dal personale aziendale.

Tipologie di Test

- **Host Assessment** – Una analisi di questo tipo punta a verificare le vulnerabilità a livello di singolo sistema. Possono ricadere in questi test i problemi di permessi, la mancanza di patch e di aggiornamenti, la non conformità ad alcune politiche di sicurezza nonché la presenza di Virus, Trojan Horse, Keylogger e Backdoors.

- **Network Assessment** – Un test di network è teso a individuare le risorse in uso nella rete, le loro vulnerabilità, la loro esposizione a potenziali minacce.

  A differenza del test sui singoli host, queste analisi sono svolte in maniera meno specifica e puntuale e quindi possono peccare nell'individuazione di alcune specifiche vulnerabilità a livello di singola macchina (soprattutto non si riesce facilmente ad indentificare un keylogger o un rootkit attraverso questo approccio).

L'assessment

- Indipendentemente dalla tipologia di assessment da svolgere il Tester dovrà strutturare la strategia di Assessment. Essa è suddivisa in:

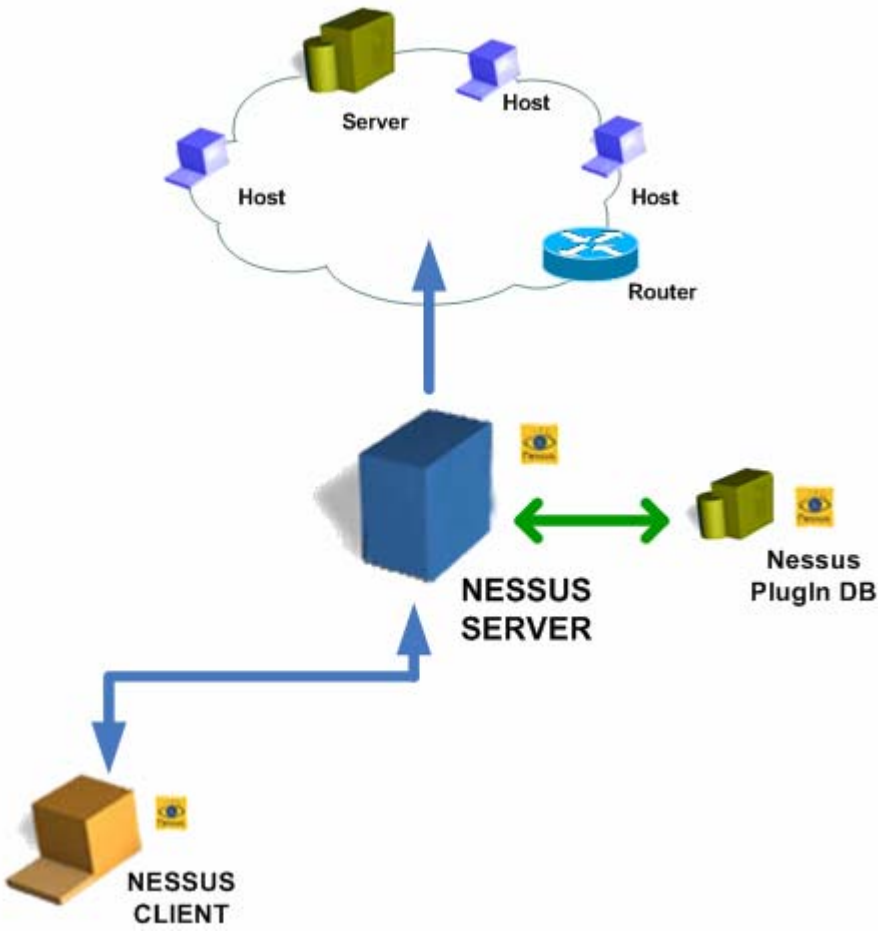
  - **Individuazione del Target**
  - **Enumerazione dei Servizi**
  - **Identificazione dei Servizi**
  - **Identificazione degli Applicativi**
  - **Identificazione delle Vulnerabilità**
  - **Report**

Cos'è Nessus

- Nessus è un vulnerability scanner.
- Nessus si occupa di analisi di vulnerabilità sia in ambiente di rete locale che in ambiente remoto.
- Tenablesecurity.com è l'azienda che lo commercializza e ne mantiene il supporto commerciale.
- E' offerto sia in versione Free (con licenza Open Source), sia in versione commerciale (con un costo legato al supporto e alla frequenza di aggiornamento).

- **Grazie a Nessus è possibile svolgere una puntuale e customizzata analisi delle vulnerabilità**

Architettura Applicativa



Architettura

- Nessus Server

- Nessus Client

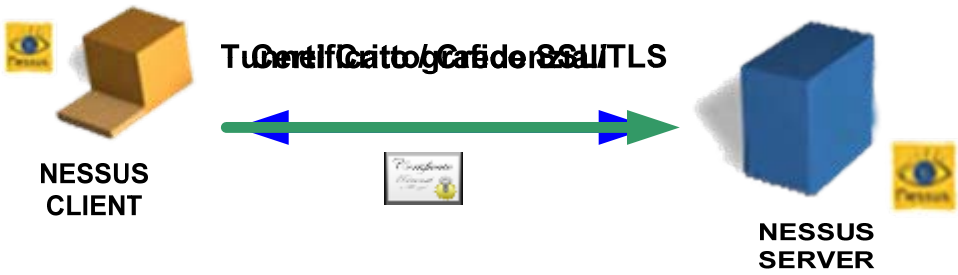
- Nessus Plug-ins.

Architetture di Analisi

- In base al suo modello funzionale Nessus può essere installato in varie modalità, anche distribuite e può essere gestito da uno o più client nella rete.
- In alcuni deployment è utile integrare Nessus con altri applicativi di monitoraggio e reportistica Open Source.
- In effetti tra i vantaggi di questo applicativo si possono citare:
    - **La flessibilità**
    - **La scalabilità**
    - **La configurabilità**

Comunicazione tra Client e Server

- **La comunicazione tra Client e Server può utilizzare sia la crittografia che l'autenticazione.**



I Plug-In

- Un aspetto molto importante di Nessus e che lo rende preferibile agli altri sistemi di analisi è il potente NASL: Nessus Attack Scripting Language o anche detto Nessus Advanced Scripting Language.
- NASL permette ai Pen-Tester di creare i propri plug-in per i vulnerability assessment.
- Il risultato è una potente sinergia di tool già pronti, forniti dal software, ampliati dai suoi servizi di aggiornamento, e gli script “fatti in casa” che permettono, a chi vuole di adoprarsi alla creazione o all’adattamento delle Proof of Concept o del nuovo fiammante “zero-day”, di allargare la propria base di test e di aumentare le casistiche di analisi.

- **Ma quanto è complesso utilizzare il NASL?**
- **Perché introdurre un nuovo linguaggio di programmazione invece di usare un linguaggio già noto come C++, Perl o Python?**

I Plug-In

- Facciamo rispondere Renaud Deraison, il creatore di Nessus.

  - **“ […]Né C++, né Perl o Python sono sicuri di per sé. Il linguaggio NASL è stato progettato per lavorare attraverso il motore Nessus e non ha la capacità di leggere dati dalla console o di interagire con il sistema operativo dell’host […]”**

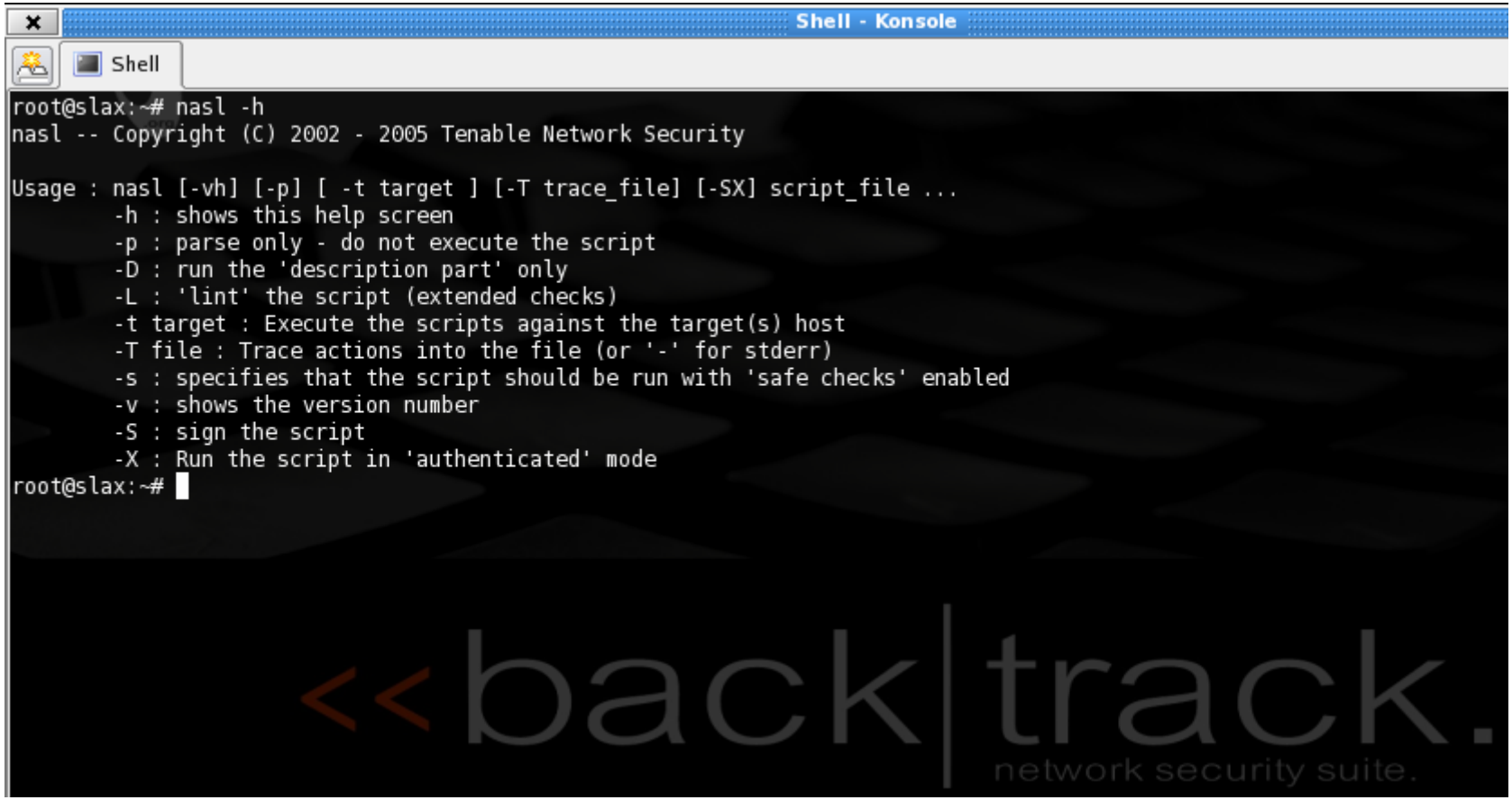
- il linguaggio NASL ha, tra i presupposti sui quali è fondato, quello di impedire che, ad esempio, predisponendo un test con un trojan si possano introdurre, per errori di programmazione, dati sensibili propri nella comunicazione tra la macchina Nessus da noi utilizzata e la macchina bersaglio.

  - **Gli script per Nessus (plug-in) girano in porzioni molto ridotte di memoria perché il linguaggio è stato ottimizzato in modo da non consumare a vuoto cicli di processore e grandi quantità di RAM.**

I Plug-In

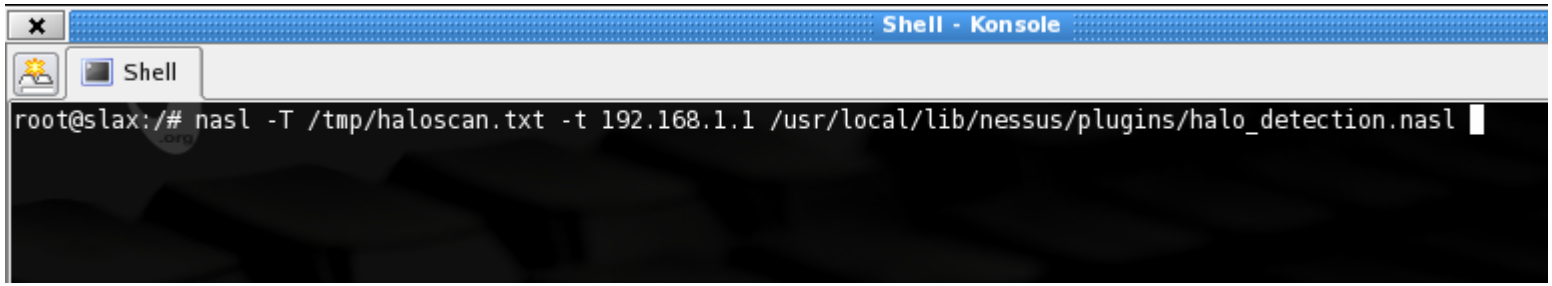
- È possibile sviluppare plug-in in linguaggi di programmazione tradizionali, ma è frequente poi il problema del dover ricompilarli e di doverne alterare le parti per farli caricare in modo corretto nel Db di Nessus.
- Il NASL è disponibile attualmente solo per sistemi Unix.
- Una volta sviluppato, il plugin potrà essere eseguito come script inserendolo nella directory apposita:
- ***/usr/local/lib/nessus/plugins/***
- La stessa directory è il repository generale di tutti i plugin, anche di quelli direttamente disponibili all'installazione del programma (lato Server).

La Console


Shell - Konsole
Shell
root@slax:~# nasl -h
nasl -- Copyright (C) 2002 - 2005 Tenable Network Security
Usage : nasl [-vh] [-p] [-t target] [-T trace_file] [-SX] script_file ...
-h : shows this help screen
-p : parse only - do not execute the script
-D : run the 'description part' only
-L : 'lint' the script (extended checks)
-t target : Execute the scripts against the target(s) host
-T file : Trace actions into the file (or '-' for stderr)
-s : specifies that the script should be run with 'safe checks' enabled
-v : shows the version number
-S : sign the script
-X : Run the script in 'authenticated' mode
root@slax:~#
<<back|track.
network security suite.

La Console: lanciare un Plug-In

- Lanciamo un Plug-In direttamente dalla console nasl



- Questo innocuo Plug-In scriverà, una volta eseguito, il risultato del suo test nel file haloscan.txt all'interno della directory /tmp
- L'host bersaglio è: 192.168.1.1
- Il percorso /usr/local/lib/nessus/plugins/halo_detection.nasl occorre per indicare al nasl dove trovare il Plug-In da utilizzare, differenziato per nome.
- Questo script nasl, innocuo come detto, è progettato per svolgere la discovery di un Server HALO (un gioco multilayer per PC) ad uno specifico indirizzo di rete, nel nostro caso 192.168.1.1

Il Plug-In: halo_detection.nasl

```
#
# Copyright (C) 2004 Tenable Network Security
#

if(description)
{
  script_id(12117);
  script_version ("$Revision: 1.5 $");

  name["english"] = "HALO Network Server Detection";
  script_name(english:name["english"]);

  desc["english"] = "
Synopsis :

A game server has been detected on the remote host.

Description :

The remote host is running a version of HALO Network Server.
The Server is used to host Internet and Local Area Network (LAN)
games.

Make sure that the use of this program is done in accordance with your
corporate security policy.

Solution :

If this service is not needed, disable it or filter incoming traffic
to this port.

Risk factor :

None";


  script_description(english:desc["english"]);
  summary["english"] = "Detects HALO Tournament Server";
  script_summary(english:summary["english"]);
  script_category(ACT_GATHER_INFO);

  script_copyright(english:"This script is Copyright (C) 2004 Tenable Network Security");
```

Il Plug-In: halo_detection.nasl

```
  script_copyright(english:"This script is Copyright (C) 2004 Tenable Network Security");

  script_family(english:"Service detection");
  script_require_keys("Settings/ThoroughTests");
  exit(0);
}


include("global_settings.inc");
if ( ! thorough_tests ) exit(0);
# start script
port = 2302;

sock = open_sock_udp(port);
if ( ! sock ) exit(0);

send (socket:sock, data:raw_string(0x5C, 0x73, 0x74, 0x61, 0x74, 0x75, 0x73, 0x5C) );

r = recv(socket:sock, length:512, timeout:3);

if ( ! r ) exit(0);

# OK, there are two modes...mode 1 is when the server is actively serving up a game
# in which case you'll get a long verbose reply from the server
# in mode 2, the server is in IDLE state and is not actively serving a game
# in mode 2, the server will just send back a quick 5 byte error msg to client

# mode 1
if (egrep(string:r, pattern:"hostname.*gamever.*maxplayers")) {
    security_note(port);
}

# mode 2
if ( (strlen(r) == 5) && (ord(r[0]) == 0xFE) && (ord(r[0]) == 0xFE) ) security_note(port);
```

Il Linguaggio NASL

- La struttura del linguaggio NASL è molto vicina a quella di altri linguaggi di programmazione
- Il NASL è disponibile attualmente solo per sistemi Unix.
- Una volta sviluppato, il plugin potrà essere eseguito come script inserendolo nella directory apposita:
- ***/usr/local/lib/nessus/plugins/***
- La stessa directory è il repository generale di tutti i plugin, anche di quelli direttamente disponibili all'installazione del programma (lato Server).

I Numeri

- NASL permette di esprimere le variabili con valori interi. E' in oltre possibile assegnare valori numerici alle variabili utilizzando rappresentazioni esadecimali.
- I Valori Esadecimali in NASL devono essere rappresentati utilizzando il prefisso "0x".
- Per esempio il numero esadecimale 0x1b corrisponde al valore 27 in base decimale.
- Inseriamo il seguente script in un file:

```
b=0x1b;
display ("il valore di b è ",b,"\n");
```

- Adesso lanciamolo utilizzando l'interprete NASL per vedere il risultato:

```
[notroot]$ nasl hex.nasl
Il valore di b è 27
```

- E' in oltre possibile inserire valori numerici in Ottale utilizzando il prefisso "0". Nel seguente esempio "x" e "y" sono equivalenti:

```
x=014; #octal
y=12; #decimal
```

Array

- Un array è un insieme di numeri o stringhe che possono essere indicate utilizzando scritture numeriche. Consideriamo il seguente script NASL:

```
myarray=make_list(1,"due");
display("Il valore del primo oggetto è ",myarray[0]," \n");
display("Il valore del secondo oggetto è ",myarray[1]," \n");
```

- Questo è il risultato delle script quando viene eseguito:

```
Il valore del primo oggetto è 1
Il valore del secondo oggetto è due
```

Variabili Locali e Globali

- Le variabili possono esistere solo all'interno di un blocco in cui sono definite.
- Un blocco è un'insieme di definizioni racchiuso da function calls.
- Per esempio se definiamo una variabile all'interno di una function call, essa non esisterà quando la function call terminerà.
- Le variabili sono locali per default.
- Nei casi in cui sia necessario invece indicare una variabile che debba esistere globalmente si dovrà utilizzare una definizione **global_var** come ad esempio:

```
global_var lamiavariabileglobale;
```

Stringhe

- NASL offre una vasta libreria di funzioni per la manipolazione di stringhe.
- Queste funzioni sono molto utili in considerazione del fatto che quando si svolge una scansione in cerca di vulnerabilità è molto probabile avere in risposta dei dati di ritorno sotto forma di stringhe.
- A questo si aggiungano tutti gli scenari nei quali è possibile inserire un input sotto forma di caratteri o voci numeriche (stringhe malformate, ad esempio) o nell'ambito della gestione dei risultati in fase di report.
- Da questi casi si comprende facilmente quanto sia importante avere una buona conoscenza di questi operatori.
- Vediamone qualcuno…

Stringhe: description

- Operatore utile per inserire una descrizione del plugin creato. Fondamentale per definire e permettere la catalogazione del plug-in.

```
If (description)
{
script_id(99999);
script_version ("$Revision: 1.2 $");
script_name(english:"Checks for /src/passwd.inc");
desc["english"]="/src/passwd.inc is usually installed by XYZ web
```

Stringhe: Chomp, Crap, Length, Data

- La funzione **chomp()** prende come parametro la funzione nella stringa e toglie qualsiasi tabulazione, spazio o formattazione aggiuntiva.
- La funzione **crap( )** viene usata per riempire un buffer con dei caratteri specifici ripetuti.
- Le funzione **length** e **data,** usate congiuntamente sono utili per specificare la lunghezza della stringa che ci si attende come risultato e il carattere che ci si attende come risultato. Ad esempio :

```
crap(length:10,data:'S')
```

- Il risultato di questa operazione sarà: SSSSSSSSSS.
- Se nel campo data non si specificano valori si avrà il valore X usato come default.

Variabili Globali Predefinite

TRUE e FALSE

- Alla variabile TRUE viene assegnato il valore 1. Alla variabile FALSE viene assegnato un valore 0.

NULL

- Questa variabile indica un valore non definito. Se una variabile di numero intero è esaminata (esempio: : i == NULL) con NULL, in primo luogo sarà paragonata a 0. Se una variabile della stringa è esaminata (esempio: str == NULL) con NULL, sarà paragonata al "" vuoto della stringa.

VARIABILI DI SCRIPT

- Ogni NASL deve specificare una singola categoria appartenete allo script_category (). Per esempio, un collegamento di cui lo scopo principale è verificare una denial-of-service vulnerability dovrebbe avere tale script_category () come segue:

```
script_category(ACT_DENIAL);
```

- Si può sostenere la funzione script_category ( ) con qualsiasi categoria come parametro. Di seguito ne elenchiamo qualcuna:
  - ACT_ATTACK;
  - ACT_DENIAL
  - ACT_DESTRUCTIVE_ATTACK
  - ACT_GATHER_INFO
  - ACT_INIT
  - ACT_SCANNER

Operatori

- Il NASL prevede confronti e assegnazione di operatori matematici. Questi operatori sono:

- Operatori Aritmetici: +, -, *, /, %, **, ++, --
- Operatori Comparativi: >, >=, <, <=, ==, !=, ><, >!<,
- Operatori per Assegnazione: =, =+, -=, *=, /=, %=,

If… else

- E' possibile utilizzare i comandi if...else per eseguire un blocco di codice in fuzione di una condizione. Per esempio, supponiamo di voler come valore della variabile port_open "1" se il valore della variabile success è positivo altrimenti, se negativo, il valore "-1" ecco un esempio:

```
if (success>0)
{
port_open=1;
}
else
{
port_open=-1;
}
```

Loop

- I loop (cicli) sono utilizzati per ripetere una parte di codice in base ad un insieme di condizioni. Di seguito verranno illustrati i diversi tipi di loop supportati dal NASL

For

- Il for (prevede tre campi separati da punto e virgola. Il primo campo è eseguito per primo e soltanto una volta. E' frequentemente utilizzato per assegnare un valore alla variabile che viene utilizzata dal loop per effettuare il ciclo. Il secondo campo è una condizione che se vera fa ripetere il ciclo. Il terzo campo viene utilizzato dal loop nel caso in cui il risultato della condizione del secondo campo è vera, incrementando o decrementando la variabile.

```
for(i=0; i < max_index(myports); i++)
{
display(myports[i],"\n");
}
```

Loop

Foreach

- Possiamo utilizzare foreach per ogni elemento di un array. Questo è utile in caso si debba ripetere un loop all'interno di un array.

Repeat...until

- Significa “ripeti...fino a che” la funzione si ripete fin quando non si ottiene un determinato risultato dopo l’esecuzione del loop. Per esempio:
- i=0;
- repeat
- {
- display ("Looping!\n");
- } until (i == 0);

While

- Un while attende la verifica di una condizione e continua il loop fino a quando tale condizione è vera. Per esempio consideriamo un while che stampi i valori interi da 1 a 10:
- i=1;
- while(i <= 10)
- {
- display(i, "\n");
- i++;
- }

Funzioni

- Una funzione è una parte di un codice che effettua un calcolo particolare. Alle funzioni possono essere attribuiti parametri di ingresso e restituire un singolo valore.
- Le funzioni possono usare gli allineamenti per restituire valori multipli.
- La seguente funzione necessita come input il valore della porta espressa in un numero intero.
- La funzione restituisce 1 se il valore è pari, 0 se dispari:

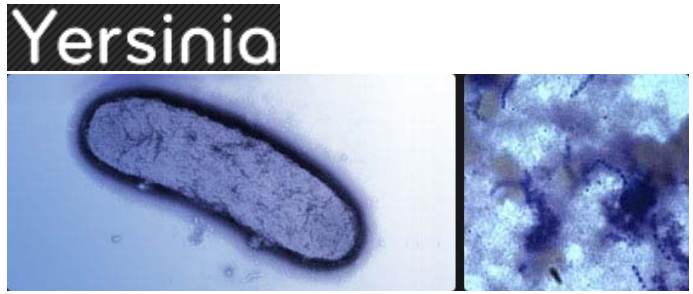
```
function is_even (port)
{
return (!(port%2));
}
```

- La funzione **is_even** realizza il calcolo del modulo per ottenere il resto quando la porta è divisa per 2.
- Se il risultato del modulo restituisce 0, il valore della porta deve essere pari. Se il risultato del modulo restituisce 1, il valore della porta deve dispari.
- L'operatore è usato per invertire il valore e questo induce la funzione a restituire 1 quando il valore del modulo risulta a 0 e 0 quando il valore del modulo risulta a 1.
- Le funzioni in NASL non si preoccupano dell'ordine dei parametri. Per passare un parametro ad una funzione, bisogna precedere con il nome del parametro: per esempio - is_even(port:22)


metasploit
"You'll Pwn Great, I Guarantee It!"
Framework 3.0 (WIN32 - UNIX)
Framework 2.7 (WIN32 - UNIX)


Nessus


back|track 2


Yersinia

GRAZIE PER L'ATTENZIONE

Stefano Maccaglia

stefano.maccaglia@gmail.com

Bibliografia e sitografia :

- nessus.org/documentation

- ***Writing security tools and exploit***
  di James C.Foster e Vincent T. Liu edito da Syngress (2006)
- ***Nessus Network Auditing***
  di HD Moore, Jay Beale, Haroon Meer, Roelof Temmingh, Charl Van Der Walt, Renaud Deraison. Edito da Syngress (2006)
- ***Nessus, Snort, & Ethereal Power Tools: Customizing Open Source Security Applications***
  di Brian Caswell, Gilbert Ramirez, Jay Beale, Noam Rathaus, Neil Archibald. Edito da Syngress (2005)