Introducción al Cracking en Linux 3 – Instalación Radare.

Programa:	**Linux CrackMe1.**
Descripción:	Continuamos con el cracking en linux.
Dificultad:	Bajísima.
Herramientas:	**Radare by pancake.**
Objetivos:	Instalación de esta completa herramienta enfocada a la ingeniería inversa..

Cracker: [Juan Jose]	Fecha: 25/04/2009

Introducción:

Bueno después de mas de dos años desde mi última introducción al Cracking en Linux, aquí he vuelto de la mano de una nueva herramienta, **Radare;** me ha parecido tan interesante que me he propuesto explicar mis avances con su instalación y manejo.

Para los que quieran recordar algo de lo que ya hemos hablado, aquí os dejo los enlaces para bajar los anteriores tutoriales:

http://www.4shared.com/file/99078428/975d6a1e/Cracking_en_Linux_1_Introduccion_.html

http://www.4shared.com/file/99078463/64e37692/Cracking_en_Linux__2-_GDB.html

También recomiendo la lectura del pdf sobre radare que hay en su web, es muy completo y aunque en ingles, toda la información que voy a comentar, la podéis completar e incluso corregir con este texto:

http://www.radare.org/get/radare.pdf

Respecto a **Radare**, he de comentar que es una herramienta de linea de comando, que en su origen según comenta su autor, **pancake**, era un editor hexadecimal que permitiera indexar con offsets de

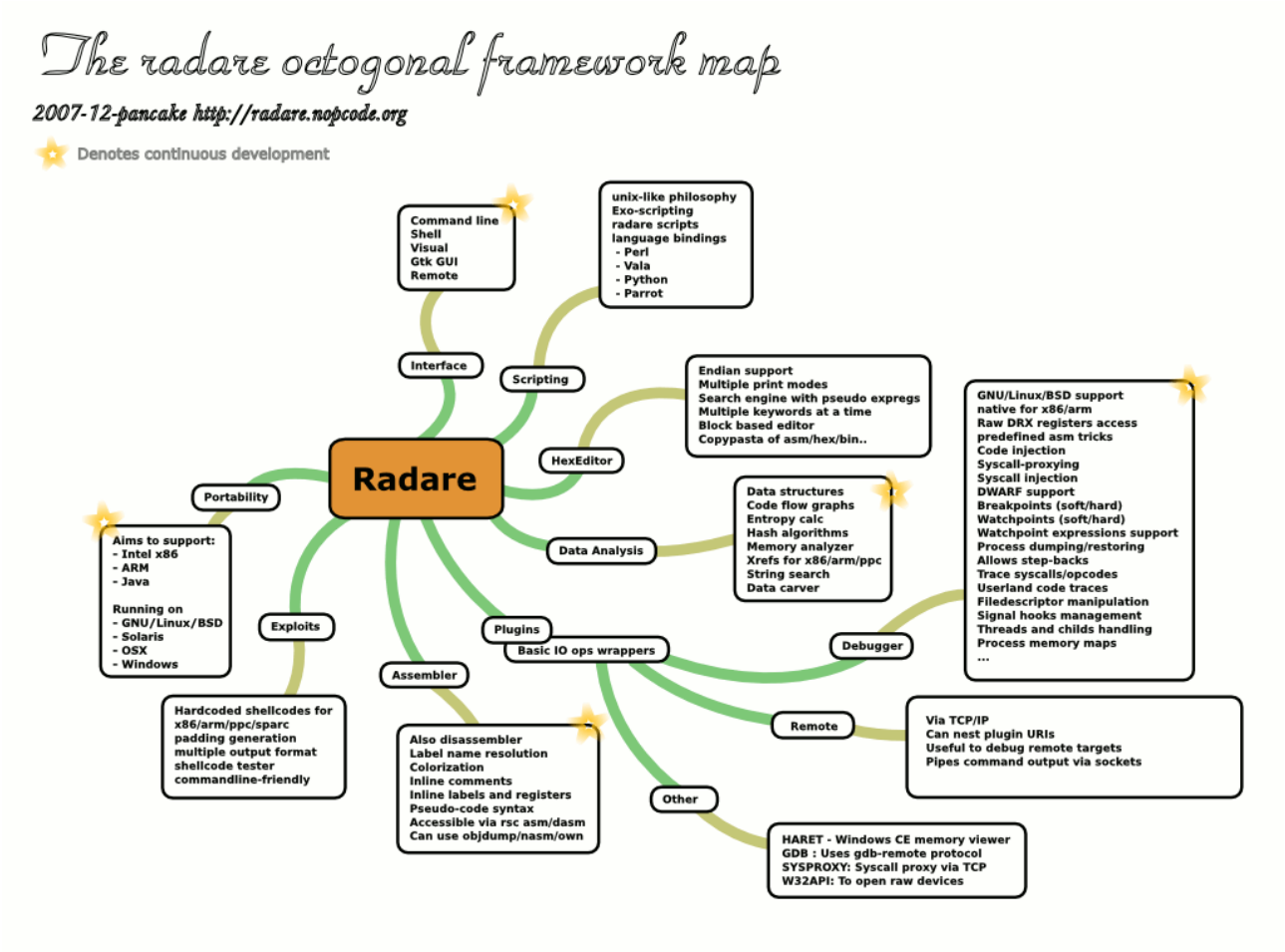
64bits, hacer busquedas, revisar hits y dumpear los resultados, de ahí su nombre:

RAw DAta REcovery

Posteriormente se ha convertido en una suite de herramientas que te dan una shell completa para la ingeniería inversa y que esta formada por varias utilidades:

- **radare** editor hexadecimal en linea de comando con varios plugins que le permite ampliar sus posibilidades.
- **rabin** obtiene información de archivos ELF / MZ / PE / CLASS archivos
- **radiff** ofrece diferentes funcionalidades para comparar binarios.
- **rasc** generador de shellcodes.
- **rasm** ensamblador y desensamblador en linea de comando.
- **xrefs** encuentra referencias cruzadas en archivos raw en, ppc, arm y x86.
- **rahash** calcula algoritmos de encriptación en archivo, bloque de datos e incluso en flujo de dastos.
- **rsc** es el lenguaje de script de radare.
- **javasm** minimalista ensamblador / desensamblador / classdumper de java.
- **armasm** minimalista ensamblador de arm.
- **rax** convierte números en diferentes bases.

Hay una imagen muy ilustrativa de sus posibilidades:



De todas estas utilidades nos vamos a centrar en **radare**, pues nos da una shell que nos permitirá utilizar tanto las demás herramientas como cualquier orden de la Shell de linux, **bash**, que podamos necesitar. Todo esto es posible gracias a la gran listas de plugins de IO que van compilados estaticos dentro del binario y que podemos ver con esta orden:

```
$ radare -L
haret       Read WCE memory ( haret://host:port )
shm         shared memory ( shm://key )
mmap        memory mapped device ( mmap://file )
serial      serial port access ( serial://path/to/dev:speed )
debug       Debugs or attach to a process ( dbg://file or pid://PID )
ewf         EnCase EWF file support ( ewf:// )
malloc      memory allocation ( malloc://size )
remote      TCP IO ( listen://:port or connect://host:port )
winedbg     Wine Debugger interface ( winedbg://program.exe )
socket      socket stream access ( socket://host:port )
gxemul      GxEmul Debugger interface ( gxemul://program.arm )
bfdbg       brainfuck debugger
gdbwrap     Connect to remote GDB using eresi's gdbwrap (gdbwrap://host:port)
gdb         Debugs/attach with gdb (gdb://file, gdb://PID, gdb://host:port)
gdbx        GDB shell interface 'gdbx://program.exe args' )
posix       plain posix file access
```

Hay otros plugins, llamados **hacks**, dentro del programa, que son ayudas para diferentes acciones en el código, estos plugins se ven desde la shell de radare con la orden **H**:

```
[0xB7EF2810]> H
01 no   nop one opcode
02 scriptedit   Script editor in GTK
03 hello        Hello hack example
04 python       python plugin
05 gtk-hello    GTK hello hack example
06 nj   Negate jump
07 lua  lua plugin
08 gtk-hello    GTK hello hack example
09 gtk-prefs    GTK preferences menu
10 gtk-actions GTK actions dialog
11 ruby         ruby plugin
12 python       python plugin
13 fj    Force jump
14 hello        Hello hack example
15 scriptedit   Script editor in GTK
16 gtk-topbar  GTK top bar (entry cmd and arch/view)
```

De esta manera podéis ver que las posibilidades son muchas, aunque yo de momento me centraré en análisis de archivos **ELF**, que son los ejecutables en GNU/Linux.

En este caso utilizaremos **radare v.1.2.2**, que es la versión estable actual; aunque también se esta desarrollando **radare v.2**, que promete ser mas rápida y completa, con un enfoque mas modular y donde **pancake** esta reescribiendo el código de cero.

Instalación.

Este programa podéis encontrarlo tanto en código fuente como en varios formatos de binarios, como podéis ver en su página:

http://www.radare.org/new/?down

Llama la atención que hay binarios para diferentes distribuciones **GNU/Linux**, para **Windows** y hasta para **Itouch/Iphone**. De todas maneras todo el tute lo estoy realizando desde **Debian testing** (actualmente se llama **squeeze**), tenedlo en cuenta, pues lo que voy a explicar puede variar en otras distribuciones, aunque en lo esencial debe coincidir.

Mi primera recomendación es que si solo quieres echarle un vistazo puedes utilizar los binarios, en mi caso el paquete **radare_20090126-1_i386.deb,** que se instala fácilmente:

```
$ su                                    ;debes ser  root para instalarlo.
Contraseña:
# dpkg -i radare_20090126-1_i386.deb
```

De todos modos, ya mismo **radare** estará en los repositorios de **debian**, así que este método sera obsoleto pues tendremos la versión mas moderna con la orden:

```
# aptitude install radare
```

Y se actualizará con el sistema cada vez que sea necesario.

De todos modos, cuando profundizas ves que el programa, instalado con el paquete **.deb,** no acaba de funcionar todo lo bien que quisieras y daba algunos errores; por lo que me decidí a compilarlo y en este caso, para obtener el código, me propuse probar la opción que recomienda **pancake**, utilizar el programa **Mercurial** para tener una imagen actualizada del código fuente en nuestro ordenador. Con mercurial conseguimos clonar el repositorio de radare y desde allí podemos instalar el programa y además lo tendremos actualizado con la última versión en muy pocos pasos.

Para empezar hay que instalar el paquete **Mercurial**, que esta en los repositorios de **Debian**:

```
$ su
Contraseña:
# aptitude install mercurial
```

Así de fácil, por cierto parece que dentro de poco, radare puede entrar a formar parte de los repositorios de **Debian**, y solo hará falta un "**aptitude install radare**" para tenerlo funcionando.

Seguimos, ya tenemos Mercurial instalado, para usarlo el comando es **hg** (ya sabéis el símbolo del mercurio, veis lo que se aprende con linux,jejeje), podemos saber sus opciones con unas ordenes básicas para sobrevivir en linux (muy útiles para los que como yo tenemos poca RAM):

```
$ man hg      ;man sirve para cualquier orden
              ó
$ hg help     ;especifico de este programa
```

Primero hay que crear un clon del lugar donde tenemos el codigo fuente, para ello ejecutamos en nuestra **Home** este comando:

juanjo@cvtucan:~$ **pwd** ;nos aseguramos que estamos en home
/home/juanjo ;¡¡Recordad tenéis que sustituir juanjo por vuestro nombre en el resto del tute!!
juanjo@cvtucan:~$ **hg clone http://radare.org/hg/radare**

Tardará un poco en descargar todo, pero el resultado es que tendremos una carpeta **/home/juanjo/radare** que será una imagen de la de internet.

Para actualizarla, debemos entrar en la carpeta **/home/juanjo/radare** y desde allí ejecutaremos estos comandos:

juanjo@cvtucan:~/radare$ **pwd** ;confirmamos donde estamos
/home/juanjo/radare
juanjo@cvtucan:~/radare$ **hg pull** ;vemos si hay actualizaciones
pulling from http://radare.org/hg/radare
searching for changes
adding changesets
adding manifests
adding file changes
added 4 changesets with 8 changes to 6 files
(run 'hg update' to get a working copy) ;nos recomienda nuestro siguiente paso
juanjo@cvtucan:~/radare$ **hg update** ;actualizamos nuestra imagen con la versión mas actual
6 files updated, 0 files merged, 0 files removed, 0 files unresolved

Así siempre tendremos la versión mas actual en esa carpeta.
Ahora llega el momento delicado de la compilación, hay dos formas, la tradicional con las ordenes clásicas:

$ **./configure --prefix=/usr** ;se configura para que los programas vayan a **/usr**
$ **make** ;se compilan los ejecutables
$ **sudo make install** ;se instala como root, tanto con la orden "**su**" como "**sudo**"

Funciona perfectamente, yo la he utilizado también y me ha dejado un programa instalado correctamente, si os da problemas **waf** esta es la segunda opción.

Hay una forma mas actual, con un script en python llamado **waf**, que es la que se recomienda y la que yo he seguido en este caso. Estando en la carpeta **/home/juanjo/radare** se ejecuta estas ordenes:

juanjo@cvtucan:~/radare$ **./waf distclean**
distclean finished successfully

Esta orden no la debéis ejecutar la primera vez, aunque de todos modos daría lo mismo, pues es la orden para borrar todo lo que se haya creado con otras compilaciones; y raro sera que , como me paso a mi , no tengáis que compilar varias veces ;-)

Ahora si, vamos al tajo, ejecutamos **waf** con al opción **configure** que nos preparara el camino para compilar y nos dirá, según lo que tengamos instalado en linux, que posibilidades tenemos y que nos falta:

juanjo@cvtucan:~/radare$ **./waf configure**

```
Checking for program gcc              : ok /usr/bin/gcc
Checking for compiler version         : ok 4.3.3
Checking for program cpp              : ok /usr/bin/cpp
Checking for program ar               : ok /usr/bin/ar
Checking for program ranlib           : ok /usr/bin/ranlib
Checking for gcc                      : ok
Checking for program g++              : ok /usr/bin/g++
Checking for compiler version         : ok 4.3.3
Checking for program ar               : ok /usr/bin/ar
Checking for program ranlib           : ok /usr/bin/ranlib
Checking for g++                      : ok
Checking for program valac            : ok /usr/bin/valac
Checking for package gthread-2.0      : ok
Checking for program version valac >= 0.1.6 : ok 0.7.0
Checking for program luac             : ok /usr/bin/luac
Checking for package glib-2.0 >= 2.10.0     : ok
Checking for package gtk+-2.0 >= 2.10.0     : ok
Checking for package vte >= 0.16          : not found
Checking for endianness               : little endian
Checking for program python           : ok /usr/bin/python
Checking for Python version >= 2.4.2  : ok 2.5.4
Checking for library python2.5        : not found
Checking for library python2.5        : not found
Checking for library python25         : not found
Checking for program python2.5-config     : not found
Checking for header Python.h          : not found
==> Use --without-python
src/plug/hack/chkruby.rb:3:in `require': no such file to load -- mkmf (LoadError)
from src/plug/hack/chkruby.rb:3
Checking for ruby mkmf                        : not found
Checking for library readline         : not found
Checking for library lua              : not found
Checking for library lua5.1           : not found
Checking for library dl               : ok
Checking for library ewf              : not found
• Prefix    : /usr
• Target    : i386-Linux
• LilEndian : True
 • HaveRuby  : False
 • EWF       : disabled
• Debugger  : enabled
• Readline  : disabled
 • SysProxy  : disabled
• GUI       : disabled
Use --without-gui : vala-waf support is not yet complete
Configuration finished successfully (00:00:03); project is now ready to build.
```

Bueno, como vemos nos faltan muchas cosas, pero tenemos lo necesario para compilar y el programa funcionará como debugger solamente ("**Debugger : enabled**"); esto en realidad puede ser suficiente pero ya que nos hemos metido a compilar vamos a intentar mejorarlo todo lo posible. Vamos a ir paso a paso:

Checking for package vte >= 0.16 : not found

En este caso nos falta el paquete **vte**, voy a comentar como solucionar estos problemas en el caso de que tengamos todo lo necesario en los repositorios; si no están, la cosa se complica, pues habría que buscarlos en internet y compilarlo también, lo cual puede traer a su vez mas problemas,uffffff... menos mal que en este caso, por lo menos en **Debian squeeze** no es necesario

En mi caso, la gran cantidad de paquetes que tiene la distribución **Debian** y el manejador de paquetes, **apt-get** o en mi caso **aptitude**, hace que podamos resolver gran parte de lo que nos falta sin muchos quebraderos de cabeza.

Primero, buscamos si hay paquetes en los repositorios que nos pueda venir bien, esto solo lo puede realizar el usuario **root**, por lo tanto con la orden **su** o **sudo** actuamos como **root:**

$ **su**
Contraseña:

aptutude update ; primero debemos estar seguros de tener los repositorios actualizados.

Ahora vamos a buscar todos los paquetes que en su nombre tengan **vte**:

aptitude search vte

p	evilvte	- an VTE based super lightweight terminal emulator
i A	libvte-common	- Terminal emulator widget for GTK+ 2.0 - common files
p	libvte-dev	- Terminal emulator widget for GTK+ 2.0 - development files
p	libvte-doc	- Terminal emulator widget for GTK+ 2.0 - documentation
i A	libvte-ruby	- VTE widget bindings for the Ruby language
i A	libvte-ruby1.8	- VTE widget bindings for the Ruby language
p	libvte0.16-cil	- CLI binding for VTE 0.16
i A	libvte9	- Terminal emulator widget for GTK+ 2.0 - runtime files
i A	python-vte	- Python bindings for the VTE widget set
v	python2.4-vte	-
v	python2.5-vte	-
p	revtex	- LaTeX documentstyle from the American Physical Society

Para entender un poco lo que nos ha salido, es bueno saber que significa las letras que aparecen al principio de cada linea, antes del paquete:

p pendiente de instalar.
i instalado, si después lleva una **A** es automatico, osea que se ha instalado con el S.O. o parte de algún paquete mas grande. Nosotros directamente no lo hemos instalado.
c se ha desinstalado el paquete pero no los archivos de configuración. Con la orden "**aptitude purge paquete**" eliminamos esos archivos.
v virtual, el paquete con ese nombre no existe aunque hace referencia a varios paquetes con un nombre parecido. Con "**aptitude show paquete**" nos puede dar el paquete real.
B paquete con dependencias rotas

Hay mas opciones (**man aptitude**) pero con estas nos apañamos.

Como vemos no hay un paquete **vte** en **Debian**, pero seguramente a lo que se refiere es **libvte-common,** para saber mas podemos usar la orden:

cvtucan:/home/juanjo# **aptitude show libvte-common**
Paquete: libvte-common
*Estado: **instalado***
Instalado automáticamente: sí
Versión: 1:0.17.4-2
Prioridad: opcional
Sección: libs
Desarrollador: Guilherme de S. Pastore <gpastore@debian.org>
Tamaño sin comprimir: 494k
Reemplaza: libvte2 (<= 0.5.1-2)
Descripción: Terminal emulator widget for GTK+ 2.0 - common files
The VTE library inserts terminal capability strings into a trie, and then uses it to determine if data received
rom a pseudo-terminal is a control sequence or just random data. The sample program "interpret" illustrates more
or less what the widget sees after it filters incoming data.

*This package contains internationalization files for the **VTE** library.*

Como veis al final estamos por buen camino, es **VTE**, y además esta instalado; pero bueno en estos casos el problema es que para compilar nos hace falta algunos archivos que contienen las funciones de cabecera, **headers**; que no son necesarios para el funcionamiento de **vte**, por lo que no están instalados, pero si son necesarios para otro programa que va a utilizar sus funciones y al compilar necesita esos archivos para añadirle esa funcionalidad. Normalmente estas funciones están en los paquetes **-dev** y a veces también en paquetes **-dbg** como vemos en este caso:

cvtucan:/home/juanjo# **aptitude show libvte-dev**
Paquete: libvte-dev
Nuevo: sí
Estado: no instalado
Instalado automáticamente: no
Versión: 1:0.17.4-2+b1
Prioridad: opcional
Sección: libdevel
Desarrollador: Guilherme de S. Pastore <gpastore@debian.org>
Tamaño sin comprimir: 1098k
Depende de: libvte9 (= 1:0.17.4-2+b1), libgtk2.0-dev (>= 2.6.0), libncurses5-dev, libxrender-dev, libxft-dev,
libfreetype6-dev, libatk1.0-dev, libpango1.0-dev, libglib2.0-dev, libsm-dev, libice-dev, zlib1g-dev,
libx11-dev, libfontconfig1-dev
Descripción: Terminal emulator widget for GTK+ 2.0 - development files
The VTE library inserts terminal capability strings into a trie, and then uses it to determine if data received
from a pseudo-terminal is a control sequence or just random data. The sample program "interpret" illustrates more
or less what the widget sees after it filters incoming data.

*This package contains **development files** for the VTE library.*

Como veis este archivo lleva los **development files**, osea los archivos de desarrollo, que seguramente es lo que buscamos, vamos a instalarlo:

aptitude install libvte-dev

Y entonces si repetimos las ordenes en la carpeta **/home/juanjo/radare**:

$ **./waf distclean**

$ ./**waf configure**

Vemos que ahora el paquete si lo ha encontrado preparado para compilar:

Checking for package vte >= 0.16 : ok

Bueno todo este rollo es para mostraros como he buscado casi todos los archivos necesarios para compilar; espero que sea una ayuda para cuando tengáis problemas con otros programas; pues aunque podría haber puesto los archivos necesarios solamente, yo prefiero enseñaros a pescar que daros los peces :-)

Seguimos sin tanto rollo viendo lo que hay que hacer para cada una de las opciones que no estaban correctas:

Checking for program python : ok /usr/bin/python
Checking for Python version >= 2.4.2 : ok 2.5.4
Checking for library python2.5 : not found
Checking for library python2.5 : not found
Checking for library python25 : not found
Checking for program python2.5-config : not found
Checking for header Python.h : not found
==> Use –without-python

Esto no puede ser, después de estudiar el curso de **Python** de **Ricardo Narvaja** no podemos consentir instalar sin tener soporte para ese genial lenguaje de programación, jeje. Como veis python está instalado en **/usr/bin/python** y el compilador necesita como mínimo la version **2.4.2**; en **debian testing** esta instalado la **v. 2.5.4**; por esa parte bien pero después faltan mas librerías, podemos intuir cuales son o si no, ya sabemos buscarlas:

aptitude install python2.5-dev python2.5-dbg ;Con esto solucionado!!

Checking for library readline : not found

Esta es fácil:

aptitude install libreadline5-dbg libreadline5-dev ;Solucionado

Checking for library lua5.1 : not found

Un poco mas complicado:

aptitude install lua5.1 liblua5.1-gtk-dev liblua5.1-0-dbg liblua5.1-0-dev ;OK

Checking for library ewf : not found

Se instala todo:

aptitude install libewf-dbg libewf-dev libewf1 ;OK

src/plug/hack/chkruby.rb:3:in `require': no such file to load -- mkmf (LoadError)
from src/plug/hack/chkruby.rb:3
Checking for ruby mkmf : not found

Este caso fue un poco especial, pues **ruby mkmf** no aparece en los repositorios ni nada parecido; tenemos que buscar con nuestro amigo **Google**, de esta manera me entere que es indicativo de la falta de los archivos de desarrollo de **ruby**, por tanto ya sabemos como solucionarlo:

aptitude install ruby1.8-dev ;OK, tampoco **ruby** podía faltar.

• GUI : disabled

Ya estaba casi todo completo pero la opción de **GUI** seguía saliendo **disabled**; como "waf configure" no me daba mas información intente el método clásico de **./configure**, que entre otras cosas me dio lo que necesitaba:

checking for gtkdialog... no
You will need gtkdialog for the gui
checking pkg-config flags for gtk+-2.0... yes
checking pkg-config flags for vte... no
Vala build disabled, ***no gtk-dev found***

Me comenta que me hace falta **gtkdialog** para la **gui**, y que tampoco encuentra **gtk-dev**. Así que instale lo necesario:

aptitude install gtkdialog libgtk-dev ; OK ----> GUI : enabled

Con todo esto, si vuelvo a repetir la **configuración**:

juanjo@cvtucan:~/radare$ **./waf distclean** ;borramos la anterior configuración
distclean finished successfully

juanjo@cvtucan:~/radare$ **./waf configure**
Checking for program gcc : ok /usr/bin/gcc
Checking for compiler version : ok 4.3.3
Checking for program cpp : ok /usr/bin/cpp
Checking for program ar : ok /usr/bin/ar
Checking for program ranlib : ok /usr/bin/ranlib
Checking for gcc : ok
Checking for program g++ : ok /usr/bin/g++
Checking for compiler version : ok 4.3.3
Checking for program ar : ok /usr/bin/ar
Checking for program ranlib : ok /usr/bin/ranlib

Checking for g++ : ok
Checking for program valac : ok /usr/bin/valac
Checking for package gthread-2.0 : ok
Checking for program version valac >= 0.1.6 : ok 0.7.0
Checking for program luac : ok /usr/bin/luac
Checking for package glib-2.0 >= 2.10.0 : ok
Checking for package gtk+-2.0 >= 2.10.0 : ok
Checking for package vte >= 0.16 : ok
Checking for endianness : little endian
Checking for program python : ok /usr/bin/python
Checking for Python version >= 2.4.2 : ok 2.5.4
Checking for library python2.5 : ok
Checking for program python2.5-config : ok /usr/bin/python2.5-config
Checking for header Python.h : ok
Checking for ruby mkmf : ok
Checking for library readline : ok
Checking for library lua : not found
Checking for library lua5.1 : ok
Checking for library dl : ok
Checking for library ewf : ok
** Prefix : /usr*
** Target : i386-Linux*
** LilEndian : True*
** HaveRuby : True*
** EWF : enabled*
** Debugger : enabled*
** Readline : enabled*
**** SysProxy : disabled***
** GUI : enabled*

Observamos que nos falta la función **SysProxy**, seguramente relacionada con el único error que tenemos:

Checking for library lua : not found

He instalado muchas librerias relacionadas con **lua** pero no hay manera :-(

De momento, con esto tenemos para empezar, vamos a **compilar** los archivos, en la carpeta **/home/juanjo/radare** debemos ejecutar :

$ **./waf**

Y por último vamos a **instalar** los archivos, como se colocarán en **/usr**, debemos tener permiso para escribir en esa carpetas, por lo que debemos ser root para terminar el proceso:

$ **su**
Contraseña:
./waf install ; si usas sudo seria **“sudo . /waf install”**

Con esto hemos terminado la instalación y podemos usar los diferentes comandos que nos proporciona **radare** en cualquier punto, pues los ejecutables se encuentran en **/usr/bin** que esta incluido en el **$PATH**.

Radare2

Aunque esta en fase de desarrollo y no acaba de funcionar muy bien, se puede tener ambos radare instalados, pues los nombres de los programas de esta versión terminan en 2, actualmente tenemos **radare2**, **rabin2**, **rahash2**, **radiff2**, **rasm2**, **rax2** y **rafind2**, este último es nuevo en la versión 2.

Para ello utilizaremos también el programa mercurial. Creamos un clon de la página donde esta el código fuente, como la vez anterior lo hacemos desde nuestra carpeta **/home** y ponemos en un terminal:

$ **hg clone http://radare.org/hg/radare2**

De esta manera se creara una carpeta en tu home llamada **/radare2** y dentro de ella podemos instalar el programa. Para ello se utiliza el método clásico:

$ **./configure --prefix=/usr** ;se configura para que los programas vayan a /usr

$ **make**

Y como root:

make install

Para utilizarlo me ha hecho falta instalar las librerías de desarrollo de **core**:

aptitude install gnome-core-devel

De momento, radare funciona de manera irregular, pero si que se ve mas rapido. Respecto al nuevo, **rafind2** sirve para buscar string y valores hexadecimales en los archivos; se ve muy interesante, ya iremos probando todo, poco a poco.

Como este rollo se ha alargado mas de lo que pensaba, continuaremos con el manejo de esta gran herramienta en la siguiente parte.

Seguimos.....
[Juan Jose]


T3aM CracksLatin0s
Juan Jose