Sociedad Andaluza para el Desarrollo de las Telecomunicaciones
CONSEJERÍA DE EMPLEO, EMPRESA Y COMERCIO

seguridad+ Y CONFIANZA DIGITAL | AndalucíaCERT CENTRO DE SEGURIDAD TIC

Informe de divulgación

Laboratorio para el análisis de malware (IV): Análisis automático

Tipo de documento:	*Informe*
Autor del documento:	*AndalucíaCERT*
Código del Documento:	*CERT-IF-8486-150806*
Edición:	*0*
Categoría	*Público*
Fecha de elaboración:	*06/08/2015*
N° de Páginas	*1 de 28*

1 TABLA DE CONTENIDOS

2 OBJETIVO Y ALCANCE

Éste es el cuarto de una serie de documentos que describen la implementación de un laboratorio para el análisis de malware. Se hablará principalmente del proceso de análisis automático de malware, usando diversas herramientas de análisis e implementando nuestra propia Cuckoo Sandbox.

3 ALCANCE

Este documento va dirigido tanto al personal de la Junta de Andalucía, como al público en general. Pretende aportar las nociones necesarias para entender y tener conocimiento sobre las técnicas que se suelen usar para combatir las amenazas de malware.

En el enfoque de este análisis se describen algunas de las herramientas que se pueden usar en plataformas GNU/Linux para realizar análisis estáticos y dinámicos de manera automática. Destacaremos la creación de un entorno Cuckoo Sandbox para obtener información mas detallada, y poder realizar análisis del malware con mas profundidad.

4 ANÁLISIS AUTOMÁTICO

El análisis de malware automático permite recoger información de un análisis estático y dinámico de forma automatizada. Se utiliza para analizar de forma masiva muestras de malware. Normalmente se usan sandboxs basadas en máquinas virtuales que permiten ejecutar software inseguro en entornos controlados, y en los que se ejecutan una serie de herramientas que obtienen gran cantidad de información del programa de forma automática. Este tipo de análisis ahorra mucho tiempo a los analistas, aunque hay que tener en cuenta algunas limitaciones:

- Las herramientas automáticas suelen ejecutar el malware sin enviarles parámetros por lo que habrá funcionalidades que no exploren.
- El malware puede detectar que se está ejecutando en una máquina virtual y reaccionar ante esto no manifestándose.
- Normalmente habrá limitaciones en el tipo de ficheros que pueda analizar la sandbox.
- Algunos malwares requieren que estén presentes ciertos registros o ficheros en el sistema y podrían no estar en la sandbox.
- Puede que el SO de una sandbox no sea el correcto para el malware.
- La sandbox no puede decirte lo que hace el malware, simplemente da un reporte del análisis. Las conclusiones las obtiene el analista o investigador.

Existen sandbox online que permiten subir un fichero para proceder a su análisis y tras esto proporcionan un reporte, se indican algunas de las más conocidas:

- Anubis: http://anubis.iseclab.org
- Malwr.com: http://malwr.com
- ThreatExpert: http://www.threatexpert.com
- Joe Sandbox: http://www.joesecurity.org

5 CUCKOO SANDBOX

Es posible implementar nuestra propia sandbox en una máquina bajo nuestro control. Esto nos permitirá una mejor integración en nuestro entorno y una mejor adaptación a nuestras necesidades personalizando su comportamiento.

Uno de los proyectos más conocidos y utilizados en estos momentos es Cuckoo Sandbox:

- http://www.cuckoosandbox.org

Con Cuckoo Sandbox se puede obtener la siguiente información:

- Trazas de llamadas a la API Win32 realizadas por procesos.
- Archivos creados, borrados y descargados durante la ejecución de un malware.
- Volcados de memoria de los procesos del malware.
- Tráfico de red generado en formato PCAP.
- Capturas de pantalla tomadas durante la ejecución del malware.
- Volcados de memoria de las máquinas virtuales.

Cuckoo puede ser utilizado para analizar los siguientes tipos de archivos:

- Ejecutables de Windows.
- Archivos DLL.
- Documentos PDF.
- Documentos Microsoft Office.

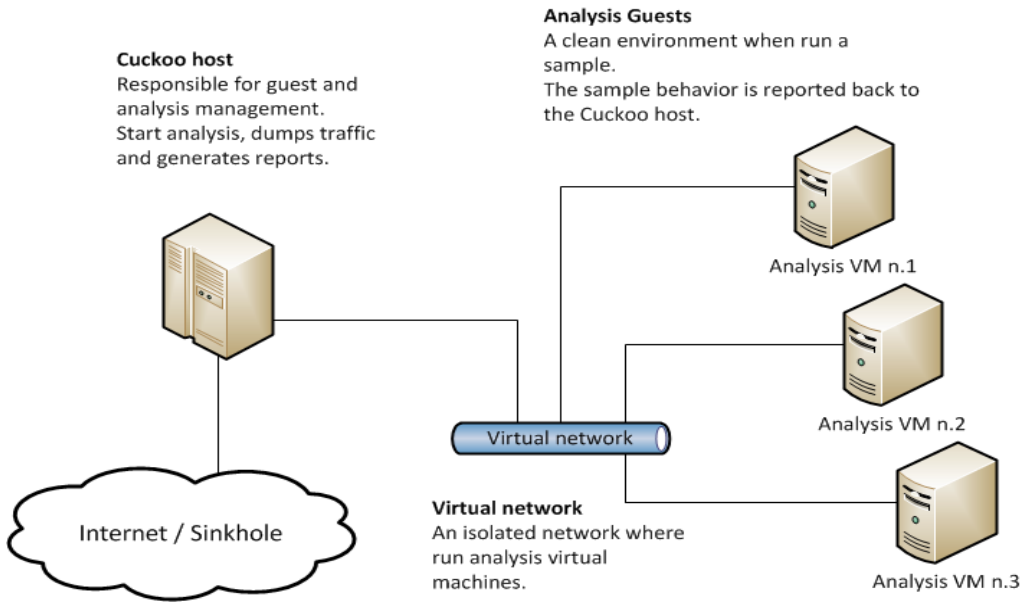
- URLs y archivos HTML.
- Scripts PHP.
- Script Visual Basic (VB).
- Archivos ZIP.
- Archivos Java JAR.

A continuación se describe la instalación, configuración y uso de Cuckoo Sandbox para usarlo en nuestro laboratorio de análisis de malware.

5.1 ARQUITECTURA

Cuckoo maneja la ejecución de muestras de malware en máquinas virtuales para su análisis. Cada análisis se lanza en una máquina virtual en estado limpio. La arquitectura está compuesta por una máquina nativa en la que está instalada Cuckoo, y una o varias máquinas virtuales que se usarán para instalar y analizar las muestras de malware. Después de realizar el análisis y obtenerse toda la información, se recupera el estado inicial de la máquina virtual.

El siguiente diagrama muestra la arquitectura de Cuckoo:



5.2 INSTALACIÓN Y CONFIGURACIÓN

Este punto explica la instalación de Cuckoo Sandbox en una máquina Linux. Sobre esta máquina se ejecutará una máquina virtual con sistema operativo Windows XP en la que se ejecutarán los análisis.

5.2.1 Preparando el host

Aunque puede ejecutarse sobre otros sistemas, inicialmente Cuckoo está pensado para ejecutarse en una máquina Linux nativa. Se describe la instalación y configuración de Cuckoo en un sistema operativo Debian. Se asume que el software de virtualización VirtualBox ya está instalado en la máquina.

5.2.1.1 Configuración

Cuckoo cuenta principalmente con seis ficheros de configuración:

- **cuckoo.conf:** configuración del comportamiento general y opciones de análisis.
- **auxiliary.conf:** para activar y configurar módulos auxiliares.
- **virtualbox.conf:** configurar el software de virtualización VirtualBox.
- **memory.conf:** configuración de Volatility.
- **processing.conf:** para activar y configurar módulos de procesamiento.
- **reporting.conf:** para activar o desactivar formatos de reporte.

Es necesario configurar como mínimo auxiliary.conf, cuckoo.conf y virtualbox.conf para hacer funcionar a Cuckoo.

5.2.2 Preparando el sistema huésped

En este punto vamos a describir cómo preparar el sistema huésped, es decir, la máquina virtual Windows en la que se ejecutará el malware.

5.2.2.1 Requisitos

En primer lugar habría que crear una nueva máquina virtual e instalar el sistema operativo. Se recomienda usar Windows XP, pero Cuckoo también funciona con Windows 7 con "User Access Control" desactivado.

El primer requisito una vez instalado el sistema operativo es instalar Python. Se podrá descargar de la página oficial. Se prefiere la versión 2.7.

- http://www.python.org/getit/releases/2.7.6/

También es necesario instalar la Python Image Library. Esta librería es usada para tomar las capturas de pantalla del escritorio de Windows mientras dura el análisis.

- http://www.pythonware.com/products/pil/

Dependiendo de los tipos de fichero que se quieran analizar habrá que instalar en el sistema las herramientas necesarias para ejecutar esos ficheros. Por ejemplo: navegadores web, visores PDFs, suites ofimáticas, etc. Recordar deshabilitar las actualizaciones automáticas o el chequeo de actualizaciones de cualquier software adicional.

5.2.2.2 Configuración de la red

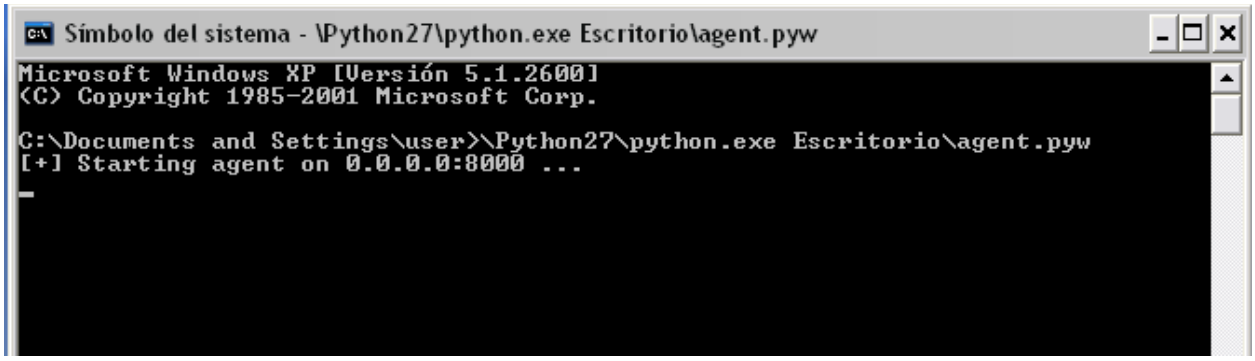
Una de las cosas más importantes es desactivar el "Firewall de Windows y las Actualizaciones Automáticas", ya que pueden afectar al comportamiento del malware bajo circunstancias normales, y pueden contaminar el análisis de la red realizada por Cuckoo, evitando que se establezcan conexiones o incluyendo peticiones irrelevantes.

La conexión a Internet de la máquina virtual se realizará usando una interfaz en modo Host-Only con el apropiado reenvío de paquetes activado y reglas de filtrado realizadas con iptables en la máquina nativa. Después de realizar la configuración de red es conveniente comprobar la conectividad entre la máquina virtual Windows y la máquina nativa.

5.2.2.3 Instalación del agente

Para la comunicación entre la máquina Windows y la nativa Cuckoo se usa un agente que se ejecuta en la máquina Windows. Éste maneja la comunicación y el intercambio de datos con la máquina nativa.

Es necesario instalar y ejecutar el agente para el correcto funcionamiento de Cuckoo. El agente se encuentra en la carpeta "agent/" de Cuckoo, el archivo "agent.py". Hay que copiar este fichero a la máquina Windows, abrir un terminal y ejecutarlo. Al ejecutarlo aparecerá una ventana. Si se quiere esconder esta ventana hay que renombrar el fichero de "agent.py" a "agent.pyw", esto evitará que la ventana se lance.



5.2.2.4 Salvando la máquina virtual

Ahora habría que salvar el estado de la máquina virtual con un snapshot. Se puede hacer el snapshot a través del entorno gráfico o mediante línea de comandos:

Igualmente, para retornar al estado conocido simplemente se restaura el snapshot.

5.3 USO

5.3.1 Arrancando Cuckoo

Para arrancar Cuckoo, ejecutamos el comando cuckoo.py que se encuentra en la raíz de la carpeta de la aplicación:

```
$ python cuckoo.py

                     __
  .----..--.--..----.|  |--..-----..-----.
  |  __||  |  ||  __||     <|  _  ||  _  |
  |____||_____||____||__|__||_____||_____|

 Cuckoo Sandbox 1.0
 www.cuckoosandbox.org
 Copyright (c) 2010-2014

 Checking for updates...
 Good! You have the latest version available.

2014-02-02 19:58:41,276 [lib.cuckoo.core.scheduler] INFO: Using "virtualbox" machine
manager
2014-02-02 19:58:44,499 [lib.cuckoo.core.scheduler] INFO: Loaded 1 machine/s
2014-02-02 19:58:44,500 [lib.cuckoo.core.scheduler] INFO: Waiting for analysis
tasks...
```

Cuckoo.py tiene las siguientes opciones:

```
$ python cuckoo.py -h
```

```
...
usage: cuckoo.py [-h] [-q] [-d] [-v] [-a]

optional arguments:
  -h, --help     show this help message and exit
  -q, --quiet    Display only error messages
  -d, --debug    Display debug messages
  -v, --version  show program's version number and exit
  -a, --artwork  Show artwork
```

Para subir un fichero y proceder a su análisis hay varias formas: línea de comandos, interfaz web, a través de las funciones python o de la API.

5.3.2 Línea de comandos

Para analizar un fichero por la línea de comandos se proporciona el comando "submit.py" que se encuentra en la carpeta "utils". Tiene las siguientes opciones disponibles:

```
# python submit.py -h
usage: submit.py [-h] [--url] [--package PACKAGE] [--custom CUSTOM]
                 [--timeout TIMEOUT] [--options OPTIONS] [--priority PRIORITY]
                 [--machine MACHINE] [--platform PLATFORM] [--memory]
                 [--enforce-timeout] [--clock CLOCK] [--tags TAGS] [--max MAX]
                 [--pattern PATTERN] [--shuffle] [--unique] [--quiet]
                 target

positional arguments:
  target                URL, path to the file or folder to analyze

optional arguments:
  -h, --help            show this help message and exit
  --url                 Specify whether the target is an URL
  --package PACKAGE     Specify an analysis package
  --custom CUSTOM       Specify any custom value
  --timeout TIMEOUT     Specify an analysis timeout
  --options OPTIONS     Specify options for the analysis package (e.g.
                        "name=value,name2=value2")
  --priority PRIORITY   Specify a priority for the analysis represented by an
                        integer
  --machine MACHINE     Specify the identifier of a machine you want to use
  --platform PLATFORM   Specify the operating system platform you want to use
                        (windows/darwin/linux)
  --memory              Enable to take a memory dump of the analysis machine
  --enforce-timeout     Enable to force the analysis to run for the full
                        timeout period
  --clock CLOCK         Set virtual machine clock
  --tags TAGS           Specify tags identifier of a machine you want to use
  --max MAX             Maximum samples to add in a row
  --pattern PATTERN     Pattern of files to submit
  --shuffle             Shuffle samples before submitting them
  --unique              Only submit new samples, ignore duplicates
  --quiet               Only print text on failure
```

Se muestran algunos ejemplos a continuación.

Entregar un fichero binario local:

```
$ ./utils/submit.py /ruta/al/fichero/binario
```

Entregar una URL:

```
$ ./utils/submit.py --url http://www.example.com
```

Entregar un fichero local y especificar una prioridad mayor:

```
$ ./utils/submit.py --priority 5 /ruta/al/fichero/binario
```

Entregar un fichero y especificar un tiempo límite para realizar el análisis:

```
$ ./utils/submit.py --timeout 60 /ruta/al/fichero/binario
```

Entregar un fichero local y especificar el tipo de empaquetado:

```
$ ./utils/submit.py --package <nombre del empaquetado> /ruta/al/fichero/binario
```

Entregar un fichero local, especificando el tipo de empaquetado y argumentos:

```
$ ./utils/submit.py --package exe --options arguments=--haceralgo /ruta/fichero
```

Entregar un fichero local para ser ejecutada en la máquina virtual cuckoo1:

```
$ ./utils/submit.py --machine cuckoo1 /ruta/fichero
```

Entregar el fichero y ejecutarlo en una máquina Windows:

```
$ ./utils/submit.py --platform windows /ruta/fichero
```

Entregar un fichero local y tomar un volcado de memoria completo de la máquina:

```
$ ./utils/submit.py --memory /ruta/fichero
```

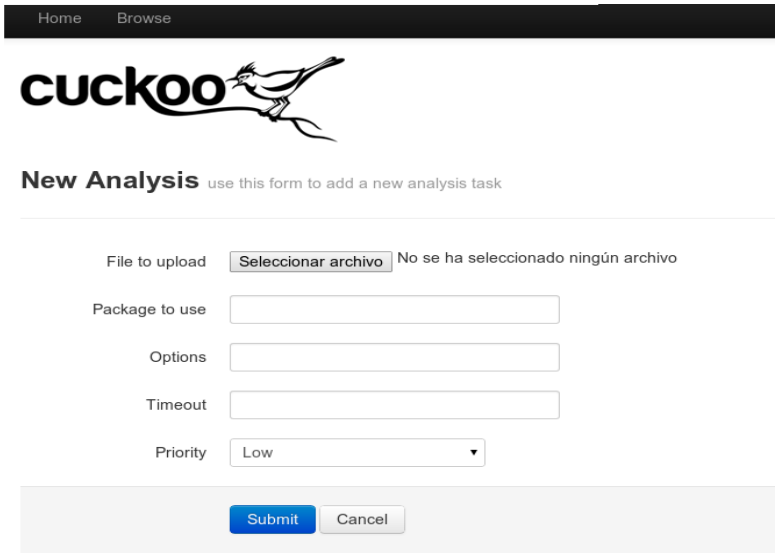
5.3.3 Interfaz web sencilla

Cuckoo proporciona una interfaz web a través de un servidor web sencillo en localhost en el puerto 8080. Mediante esta interfaz se podrá navegar por los diferentes reportes y entregar nuevos ficheros para analizar.

Para arrancar esta interfaz web se ejecuta el siguiente comando:

```
$ ./utils/web.py
Bottle server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.
```

Podremos acceder a través de un navegador web en la dirección: http://localhost:8080



5.3.4 API REST

Cuckoo proporciona una API REST implementada en Bottle.py. Para que el servicio funcione es necesario tener instalado Bottle versión 0.10 o superior.

```
$ sudo apt-get install python-bottle
```

Para arrancar el servidor API:

```
$ ./utils/api.py
```

Por defecto escuchará en localhost:8090. Se puede cambiar la IP y el puerto mediante:

```
$ ./utils/api.py --host 0.0.0.0 --port 1337
```

A continuación se muestran los recursos disponibles:

Recurso	Descripción
POST /tasks/create/file	Añade fichero para ser procesado.
POST /tasks/create/url	Añade url para ser procesada
GET /tasks/list	Devuelve una lista de tareas almacenadas en la base de datos
GET /tasks/view	Devuelve el detalle de la tarea asignada especificada por ID
GET /tasks/delete	Borra tarea de la base de datos y sus resultados
GET /tasks/report	Devuelve el reporte generado asociado a la tarea especificada por ID
GET /files/view	Busca los binarios analizados por el hash o ID
GET /files/get	Devuelve el contenido de un binario especificado por el hash
GET /machines/list	Devuelve la lista de máquinas de análisis disponibles
GET /machines/view	Devuelve el detalle de la máquina asociada con el nombre especificado
GET /cuckoo/status	Devuelve el estado básico de Cuckoo

Por ejemplo, para añadir un fichero a las tareas pendientes se podría hacer la siguiente petición, la cual devolvería el ID de la nueva tarea creada:

```
curl -F file=@/path/to/file http://localhost:8090/tasks/create/file
```

La respuesta sería:

```
{
    "task_id" : 1
}
```

Se puede consultar la documentación oficial para tener un mayor detalle de las consultas que se pueden hacer:

- http://docs.cuckoosandbox.org/en/latest/usage/api/

5.3.5 Funciones Python

Para automatizar análisis mediante scripts se puede hacer uso de las funciones add_path() y add_url().

```
add_path(file_path[, timeout=0[, package=None[, options=None[, priority=1[,
custom=None[, machine=None[, platform=None[, memory=False[,
enforce_timeout=False]]]]]]]])
```

```
add_url(url[, timeout=0[, package=None[, options=None[, priority=1[,
custom=None[,machine=None[, platform=None[, memory=False[,
enforce_timeout=False]]]]]]]])
```

Por ejemplo, para subir un fichero mediante la función add_path, ejecutamos el interprete python:

```
>>> from lib.cuckoo.core.database import Database
>>> db = Database()
>>> db.add_path("/tmp/malware.exe")
1
>>>
```

5.3.6 Interfaz web

Se proporciona una interfaz web completa realizada con Django que permite subir ficheros, navegar por los reportes y realizar búsquedas por los resultados de los análisis.

Puede ser configurada editando el fichero "local_settings.py" en la ruta web/web/:

```
# If you want to customize your cuckoo path set it here.
# CUCKOO_PATH = "/where/cuckoo/is/placed/"

# If you want to customize your cuckoo temporary upload path set it here.
# CUCKOO_FILE_UPLOAD_TEMP_DIR = "/where/web/tmp/is/placed/"

# Maximum upload size.
MAX_UPLOAD_SIZE = 26214400

# Override default secret key stored in secret_key.py
# Make this unique, and don't share it with anybody.
# SECRET_KEY = "YOUR_RANDOM_KEY"

# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# On Unix systems, a value of None will cause Django to use the same
# timezone as the operating system.
```

```
# If running in a Windows environment this must be set to the same as your
# system time zone.
TIME_ZONE = "Europe/Madrid"

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = "es-ES"

ADMINS = (
    # ("Your Name", "your_email@example.com"),
)

MANAGERS = ADMINS

# Allow verbose debug error message in case of application fault.
# It's strongly suggested to set it to False if you are serving the
# web application from a web server front-end (i.e. Apache).
DEBUG = True

# A list of strings representing the host/domain names that this Django site
# can serve.
# Values in this list can be fully qualified names (e.g. 'www.example.com').
# When DEBUG is True or when running tests, host validation is disabled; any
# host will be accepted. Thus it's usually only necessary to set it in production.
ALLOWED_HOSTS = ["*"]
```

Para arrancar esta interfaz web habría que ejecutar el siguiente comando en la ruta web/:

```
$ python manage.py runserver
```

También se puede especificar un puerto y una IP en la que escuchar:

```
$ python manage.py runserver 0.0.0.0:PORT
```

5.3.7 Análisis de paquetes

Los paquetes de análisis son el componente central de Cuckoo Sandbox. Consisten en clases estructuradas Python, las cuales son ejecutadas en las máquinas virtuales y marcan como debería de realizarse el análisis para ese tipo de paquete.

Cuckoo proporciona algunos paquetes de análisis por defecto que se pueden usar, también se puede crear uno propio modificando uno existente. Se pueden encontrar en la ruta "analyzer/windows/modules/packages/".

Los paquetes existentes son los siguientes:

- **applet:** usado para analizar Applets Java.
- **bin:** usado para analizar binarios de datos genéricos, como shellcodes.
- **cpl:** usado para analizar Control Panel Applets.
- **dll:** usado para analizar Librerías de Linkado Dinámico.
- **doc:** usado para analizar documentos Microsoft Word.
- **exe:** paquete por defecto para analizar ejecutables de Windows genéricos.
- **generic:** usado para ejecutar y analizar muestras genéricas vía cmd.exe.
- **html:** usado para analizar el comportamiento de Internet Explorer cuando abre el archivo HTML dado.
- **ie:** usado para analizar el comportamiento de Internet Explorer abriendo la URL dada.
- **jar:** usado para analizar contenedores Java JAR.
- **pdf:** usado para ejecutar y analizar documentos PDF.
- **vbs:** usado para ejecutar y analizar archivos VBScript.
- **xls:** usado para analizar documentos Microsoft Excel.
- **zip:** usado para ejecutar y analizar archivos Zip.

Se puede indicar el tipo de análisis a usar especificando su nombre en el momento de entrega del fichero:

```
$ ./utils/submit.py --package <package name> /path/to/malware
```

Por ejemplo:

```
$ ./utils/submit.py --package dll --options function=FunctionName /path/to/malware
```

5.3.8 Análisis de resultados

Cuando se completa un análisis, se generan ficheros con información de la muestra y se almacenan en directorios dentro de la ruta "storage/analyses/", dentro de un subdirectorio nombrado con un ID numérico que representa la tarea de análisis en la base de datos.

Se muestra un ejemplo de la estructura de un directorio de análisis:

```
.
|-- analysis.conf
|-- analysis.log
|-- binary
|-- dump.pcap
|-- memory.dmp
|-- files
|   |-- 1234567890
|       `-- dropped.exe
|-- logs
|   |-- 1232.raw
|   |-- 1540.raw
|   `-- 1118.raw
|-- reports
|   |-- report.html
|   |-- report.json
|   |-- report.maec-40.xml
|   `-- report.metadata.xml
`-- shots
    |-- 0001.jpg
    |-- 0002.jpg
    |-- 0003.jpg
    `-- 0004.jpg
```

- **analysis.conf:** Éste es un archivo de configuración generado por Cuckoo para instruir al analizador algunos detalles sobre el análisis actual. No tiene interés para el usuario, es usado internamente por la sandbox exclusivamente.
- **analysis.log:** Este log es generado por el analizador que contiene una traza de la ejecución del análisis dentro del entorno huésped. Reportará la creación de procesos, archivos y errores eventuales ocurridos durante la ejecución.
- **dump.pcap:** Esto es un volcado de tráfico de red generado por tcpdump o cualquier otro sniffer de red.
- **memory.dmp:** En el caso que esté activada, contiene el volcado completo de la máquina de análisis.
- **files:** Contiene todos los ficheros con los que el malware ha operado y Cuckoo fue capaz de volcar.
- **logs:** Contiene todos los logs generados por el proceso de monitorización de Cuckoo.
- **reports:** Contiene los reportes generados por Cuckoo.

- **shots:** Este directorio contiene las capturas de pantalla tomadas en la máquina huésped durante la ejecución del malware.

5.3.9 Análisis de paquetes.

Cuckoo por defecto ofrece algunos paquetes que se pueden utilizar para personalizar procedimientos de análisis dentro de los sistemas virtualizados.

Algunos ejemplos:

Análisis de ejecutables genéricos de Windows: shares/setup/packages/exe.py

```
import os
import sys

sys.path.append("\\\\VBOXSVR\\setup\\lib\\")

from cuckoo.execute import cuckoo_execute
from cuckoo.monitor import cuckoo_monitor

# The package main function "cuckoo_run" should follow a fixed structure in
# order for Cuckoo to correctly handle it and its results.
def cuckoo_run(target_path):
    # Every analysis package can retrieve a list of multiple process IDs it
    # might have generated. All processes added to this list will be added to
    # the monitored list, and Cuckoo will wait for all of the to complete their
    # execution before ending the analysis.
    pids = []

    # The following functions are used to launch a process with the simplified
    # "cuckoo_execute" function. This function takes as arguments (in specific
    # order):
    # - a path to the executable to launch
    # - arguments to be passed on execution
    # - a boolean value to specify if the process have to be created in
    #   suspended mode or not (it's recommended to set it to True if the
    #   process is supposed to be injected and monitored).
    suspended = True
    (pid, h_thread) = cuckoo_execute(target_path, None, suspended)

    # The function "cuckoo_monitor" invoke the DLL injection and resume the
    # process if it was suspended. It needs the process id and the main thread
    # handle returned by "cuckoo_execute" and the same boolean value to tell it
    # if it needs to resume the process.
```

```
33      cuckoo_monitor(pid, h_thread, suspended)
34
35      # Append all the process IDs you want to the list, and return the list.
36      pids.append(pid)
37      return pids
38
39  def cuckoo_check():
40      return True
41
42  def cuckoo_finish():
43      return True
```

Explicamos el código:

En la línea 1 y 2, importamos los módulos os y sys de Python.

En la línea 4 añadimos "*\\VBOXSVR\setup\lib*" a la lista de rutas de módulos de Python: esto nos permitirá invocar módulos de Cuckoo directamente desde la carpeta compartida.

Se definen tres funciones:

cuckoo_run ()

cuckoo_check ()

cuckoo_finish ()

En el ejemplo del paquete simplemente se ejecuta el binario situado en TARGET_PATH en modo suspendido e instruye a Cuckoo para inyectar el proceso y empezar a monitorizar.

Un ejemplo un poco más complejo es el paquete de análisis de PDF (ubicado en "*shares/setup/packages/pdf.py*"):

```
1   import os
2   import sys
3
4   sys.path.append("\\\\VBOXSVR\\setup\\lib\\")
5
6   from cuckoo.execute import cuckoo_execute
7   from cuckoo.monitor import cuckoo_monitor
8
9   def cuckoo_run(target_path):
10      pids = []
```

```
11
12      # Customize this Path with the correct one on your Windows setup.
13      adobe_reader = "C:\\Program Files\\Adobe\\Reader 9.0\\Reader\\AcroRd32.exe"
14
15      suspended = True
16      (pid, h_thread) = cuckoo_execute(adobe_reader, "\"%s\"" % target_path, suspended)
17      cuckoo_monitor(pid, h_thread, suspended)
18
19      pids.append(pid)
20      return pids
21
22  def cuckoo_check():
23      return True
24
25  def cuckoo_finish():
26      return True
```

En este ejemplo tenemos la misma estructura, con la única diferencia que en lugar de ejecutar el archivo en "TARGET_PATH", se ejecuta Adobe Reader con "TARGET_PATH" como argumento.
De esta manera, básicamente instruye a Cuckoo para monitorizar lo que Adobe Reader está haciendo al abrir el archivo PDF dado.

cuckoo_run ()

Esta función marca el punto de partida del análisis. En este punto se debe definir todas las operaciones que debe realizar como inicialización del análisis.

Esto podría incluir la ejecución de procesos, creación de archivos, la inyección de los procesos y todo lo que pueda necesitar para llevar a cabo.

Debe devolver una lista de los PID que será utilizado por Cuckoo para supervisar el estado del proceso: cuando todos los procesos supervisados completen su ejecución, Cuckoo terminará el análisis y mostrara la salida. Si no devuelve ningún proceso, Cuckoo asumirá que no hay un proceso supervisado y se acabara de ejecutar para la cantidad de segundos especificados por el tiempo de espera de análisis.

cuckoo_check ()

Esta función se ejecuta cada segundo durante el análisis. Se puede utilizar para realizar comprobaciones personalizados o cualquier otra operación necesaria.

Si la función de cuckoo_check () devuelve un False, Cuckoo supondrá que el paquete coincide con una comprobación condicional y se dará por terminado el análisis anterior.

cuckoo_finish ()

Esta función se ejecuta cuando se ha completado el análisis. Puede ser utilizado para cualquier propósito post-análisis como copiar archivos o cualquier otra operación que pueda necesitar para llevar a cabo antes de que la máquina virtual esté apagada.

5.3.10 Módulos de Cuckoo.

En este apartado se explicará como personalizar Cuckoo. Se aprovecha que la arquitectura de Cuckoo está modularizada para poder personalizarse, adaptándose así para cada usuario.

5.3.10.1 Módulos auxiliares

Los módulos auxiliares definen algunos procedimientos que necesitan ser ejecutados en paralelo en todo análisis. Todos los módulos auxiliares deben ser situados en el directorio "*modules/auxiliary/directory"*.

```
1  from lib.cuckoo.common.abstracts import Auxiliary
2
3  class MyAuxiliary(Auxiliary):
4
5      def start(self):
6          # Do something.
7
8      def stop(self):
9          # Stop the execution.
```

La función "start()" se ejecutará antes de comenzar el análisis en la máquina y ejecutará el archivo malicioso, mientras que la función "stop()" se lanzará en cada finalización de los procesos de análisis, antes de lanzar los procedimientos de presentación de informes.

Por ejemplo, un modulo auxiliar proporcionado por Cuckoo se llama "*sniffer.py"* y se encarga de ejecutar "*tcpdump"* con el fin de obtener el tráfico de red que se está generando.

5.3.10.2 Módulos del sistemas

Los módulos del sistema definen cómo Cuckoo debe interactuar con el software de virtualización. Desde que se decidió no utilizar un proveedor en concreto, a partir de la versión 0.4 es capaz de usarse la solución que se prefiera, y en el caso de que no esté soportada por defecto se puede crear un script en python personalizado para definir cómo Cuckoo debe usarlo.

Todos los módulos del sistema deben colocarse en "*modules/machinery/"*. Un módulo de este tipo podría ser como el siguiente:

```
1  from lib.cuckoo.common.abstracts import Machinery
2  from lib.cuckoo.common.exceptions import CuckooMachineError
3
4  class MyMachinery(Machinery):
5        def start(self, label):
6              try:
7                    revert(label)
8                    start(label)
9              except SomethingBadHappens as e:
10                   raise CuckooMachineError("OPS!")
11
12       def stop(self, label):
13             try:
14                   stop(label)
15             except SomethingBadHappens as e:
16                   raise CuckooMachineError("OPS!")
```

Los únicos requerimientos que exige Cuckoo son que herede de la clase "*Machinery"*, tener una función "start()" y una "stop()" y lanzar un "*CuckooMachineError"* cuando algo falle.

Como se sobreentiende, estos módulos forman parte de la configuración de Cuckoo, por lo que merece la pena asegurarse de que están bien depurados y se comporten de forma estable ante todo tipo de errores.

Los módulos comentados en este apartado deben venir con un archivo de configuración dedicado a los mismos, y situado en "*conf/<machinery_module_name>.conf"*. Por ejemplo para el módulo "*modules/machinery/kvm.py* " deberá existir el archivo "*conf/kvm.conf"*.

```
[kvm]
# Specify a comma-separated list of available machines to be used. For each
# specified ID you have to define a dedicated section containing the details
# on the respective machine. (E.g. cuckoo1,cuckoo2,cuckoo3)
machines = cuckoo1
```

```
[cuckoo1]
# Specify the label name of the current machine as specified in your
# libvirt configuration.
label = cuckoo1

# Specify the operating system platform used by current machine
# [windows/darwin/linux].
platform = windows

# Specify the IP address of the current machine. Make sure that the IP address
# is valid and that the host machine is able to reach it. If not, the analysis
# will fail.
ip = 192.168.0.10
```

5.3.10.3 Paquetes de análisis

Los paquetes de análisis están estructurados en clases (Python) y describen como los analizadores de Cuckoo deben llevar a cabo el procedimiento de análisis de un archivo determinado. Usted puede crear sus propios paquetes y agregarlos junto con los que ya se encuentran por defecto.

Por ejemplo, se podría ver el paquete por defecto para analizar ejecutables de Windows.

```
1  from lib.common.abstracts import Package
2
3  class Exe(Package):
4      """EXE analysis package."""
5
6      def start(self, path):
7          args = self.options.get("arguments")
8          return self.execute(path, args)
```

En el siguiente enlace se puede ver de forma más detallada este tipo de paquetes y qué hay que tener en cuenta para crear los suyos propios.

- https://cuckoo.readthedocs.org/en/latest/customization/packages/

5.3.10.4 Módulos de procesamiento

Los módulos de procesamiento de Cuckoo son scripts en Python que te permiten definir maneras personalizadas de analizar los resultados generados por la sandbox y anexar información a un contenedor global que se usará más tarde por las firmas y los módulos de informes.

Cada usuario se puede crear los módulos que quiera, siguiendo una estructura que se mostrará a continuación.

Una vez se ha realizado el análisis, Cuckoo invoca a los módulos de procesamiento disponibles localizados en la ruta "*/modules/processing/directory"*. Cada módulo deberá tener también una sección en el fichero "*conf/processing.conf"*. Por ejemplo, si se crea el módulo "*module/processing/foobar.py"* deberá existir una sección en el fichero "*conf/processing.conf* "como la siguiente:

```
[foobar]
enabled = on
```

Cada módulo deberá ser inicializado y ejecutado; y el resultado devuelto se anexará en la estructura de datos que se puede encontrar en el *contenedor global*.

Este contenedor es únicamente un gran diccionario en Python que incluye los resultados obtenidos por todos los módulos clasificados por su clave de identificación.

Cuckoo también proporciona un conjunto de módulos por defecto, y es importante que dichos módulos no se vean modificados, lo que provocaría que la estructura del *contenedor global* cambiase y los módulos de los informes no sean capaces de extraer la información necesaria y poder generar correctamente dichos informes.

Puede encontrarse más información acerca de este tipo de módulos en:

- https://cuckoo.readthedocs.org/en/latest/customization/processing/

5.3.10.5 Firmas

Con Cuckoo pueden crearse firmas personalizadas que se pueden ejecutar sobre los resultados de los análisis en busca de un patrón predefinido que podría representar un comportamiento malicioso en particular o algún indicador de interés.

Dichas firmas son muy útiles para simplificar la interpretación de los resultados, así como la identificación automática de muestras de malware de interés. Algunos de los ejemplos en los que se pueden utilizar firmas son:

- Identificar algún malware en particular aislando algunos comportamientos únicos.
- Detectar modificaciones interesantes que el malware realiza en el sistema, como la instalación de controladores de dispositivos.
- Identificar determinadas categorías de malware, como troyanos bancarios o ransomware aislando acciones típicas comúnmente realizados por aquellos.
- Clasificar las muestras en las categorías de malware.

Pueden encontrarse ya firmas realizadas por la comunidad en los distintos repositorios de Cuckoo. Puede encontrarse documentación sobre cómo crear firmas en el siguiente enlace:

- https://cuckoo.readthedocs.org/en/latest/customization/signatures/

5.3.10.6 Módulos de presentación de informes

Una vez que se ha procesado toda la información y se han obtenido los resultados del análisis, Cuckoo utiliza todos los módulos de reporte que tiene disponibles, que harán uso de dichos resultados y análisis y crearán diferentes tipos de informes.

Todos los módulos de informes deben ser colocados dentro del directorio *modules/reporting*. Cada módulo debe tener también una sección dedicada en el fichero *conf/reporting.conf*, por ejemplo si se crea el módulo *module/reporting/foobar.py* deberá de existir la sección correspondiente en el fichero *conf/reporting.conf*.

[foobar]
enabled = on

Para más información acerca de la creación de estos módulos, puede verse en el siguiente enlace:

- https://cuckoo.readthedocs.org/en/latest/customization/reporting/

5.3.11 Herramientas y Utilidades.

- Utilidades de borrado:

  Borra resultados de los análisis, los binarios, base de datos SQLite y registros.

```
$ ./utils/clean.sh
```

- Utilidades de procesamiento:

  Con el motor de procesamiento de resultados es posible volver a generar resultados ya guardados anteriormente en una carpeta de análisis y mediante un motor de búsqueda, puede realizar la depuración y el desarrollo.

  Ejemplo: Ejecución nuevamente del motor de informes para el análisis numero 1.

```
$ ./utils/process.py 1
```

Volvemos a generar los informes:

```
$ ./utils/process.py –report 1
```

Opciones de uso:

```
$ ./utils/process.py -h

usage: process.py [-h] [-d] [-r] [-p PARALLEL] id

positional arguments:
  id                    ID of the analysis to process (auto for continuous
                        processing of unprocessed tasks).

optional arguments:
  -h, –help            show this help message and exit
  -d, –debug           Display debug messages
  -r, –report          Re-generate report
  -p PARALLEL, –parallel PARALLEL
                        Number of parallel threads to use (auto mode only).
```

Nota: *Una buena practica si usamos muchas maquinas virtuales sería ejecutar un process.py y desactivar la opción Cuckoo reporting en cuckoo.conf para aumentar el rendimiento del sistema.*

- Utilidades de descarga:

Esta utilidad descarga firmas del repositorio de la comunidad de Cuckoo e instala módulos adicionales específicos en la configuración local, como por ejemplo la identificación de la actualización de todas las firmas mas recientes que están disponibles.

Opciones de uso:

```
$ ./utils/community.py -h

usage: community.py [-h] [-a] [-s] [-p] [-m] [-r] [-f] [-w] [-b BRANCH]
```

```
optional arguments:
  -h, –help            show this help message and exit
  -a, –all             Download everything
  -s, –signatures      Download Cuckoo signatures
  -p, –processing      Download processing modules
  -m, –machinemanagers
                       Download machine managers
  -r, –reporting       Download reporting modules
  -f, –force           Install files without confirmation
  -w, –rewrite         Rewrite existing files
  -b BRANCH, –branch BRANCH
                       Specify a different branch
```

Ejemplo: instalar todas las firmas disponibles:

```
$ ./utils/community.py  - -signatures - -force
```

- Utilidades para estadísticas:
  Esta simple utilidad muestra en pantalla algunas estadísticas sobre muestras ya procesadas:

```
$ ./utils/stats.py

1 samples in db
1 tasks in db
pending 0 tasks
running 0 tasks
completed 0 tasks
recovered 0 tasks
reported 1 tasks
failed_analysis 0 tasks
failed_processing 0 tasks
```

roughly 32 tasks an hour

roughly 778 tasks a day

- Utilidades de maquinas:

La utilidad machine.py está diseñado para automatizar la configuración de las máquinas virtuales en el Cuckoo. Toma una lista de los detalles de la máquina como argumentos y los escribe en el archivo de configuración específico *(cuckoo.conf)* .

Opciones de uso disponibles:

```
$ ./utils/machine.py -h
usage: machine.py [-h] [–debug] [–add] [–ip IP] [–platform PLATFORM]
                  [–tags TAGS] [–interface INTERFACE] [–snapshot SNAPSHOT]
                  [–resultserver RESULTSERVER]
                  vmname

positional arguments:
  vmname                Name of the Virtual Machine.

optional arguments:
  -h, –help            show this help message and exit
  –debug               Debug log in case of errors.
  –add                 Add a Virtual Machine.
  –ip IP               Static IP Address.
  –platform PLATFORM   Guest Operating System.
  –tags TAGS           Tags for this Virtual Machine.
  –interface INTERFACE
                       Sniffer interface for this machine.
  –snapshot SNAPSHOT   Specific Virtual Machine Snapshot to use.
  –resultserver RESULTSERVER
                       IP:Port of the Result Server.
```

6 CONCLUSIONES

Con las herramientas descritas en estas entregas divulgativas se puede disponer de un laboratorio para la investigación de nuevas amenazas. El resultado de las investigaciones nos podría ayudar a identificar indicadores de compromiso que nos permitan detectar la presencia en la red o en equipos de códigos maliciosos que pongan en riesgo la información de la organización.

Tenemos disponible un conjunto de herramientas simples con las que podemos realizar un análisis estático y obtener bastante información de las características de la pieza. Si queremos profundizar un poco más podemos hacer uso de herramientas más complejas, como los desensambladores, que nos aportarán aún más información.

El siguiente paso sería preparar un entorno en el que ejecutar la muestra y poder realizar un análisis dinámico de forma segura. Las máquinas virtuales nos permitirán controlar fácilmente las conexiones que realiza un malware y evitar que éste acceda a la red local, así como deshacer todas las modificaciones que ha realizado un malware en el sistema fácilmente.

El análisis automático usando sandboxs y máquinas virtuales proporcionará gran cantidad de información de una muestra de forma rápida. Herramientas como la expuesta en este documento nos permiten analizar estática y dinámicamente muestras de forma masiva. Se pueden usar estas herramientas automáticas como complemento al análisis manual, ya que este último nos permitirá realizar un análisis más particularizado y en profundidad.

7 REFERENCIAS

http://docs.cuckoosandbox.org/en/latest/