UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

Nombre del alumno: Jorge Alejandro Rodríguez Villagrá

Título de proyecto: Despliegue de redes señuelo mediante el uso de herramientas de creación de escenarios virtuales

Nombre del tutor: D. David Fernández Cambronero

Nombre de los miembros del tribunal:

Presidente: D. David Fernández Cambronero
Vocal: D. Francisco Javier Ruiz Piñar
Secretario: D. Luis Bellido Triana
Suplente: D. Francisco González Vidal

Fecha de lectura y defensa:

Calificación obtenida:

// Agradecimientos

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS

DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Despliegue de redes señuelo mediante el uso de herramientas de creación de escenarios virtuales

JORGE ALEJANDRO RODRÍGUEZ VILLAGRÁ

julio de 2012

Índice

Listado código

Listado figuras

1 Introducción

I reject your reality, and substitute my own.

Adam Savage. Mythbuster

1 Introducción

1.1 Introducción.

El objetivo de este proyecto fin de carrera ha sido el diseño y desarrollo de una serie de modificaciones a una herramienta de gestión de escenarios de red virtuales para permitir la integración de máquinas virtuales con sistema operativo Windows y routers CISCO virtualizados, todo con el objeto de aplicarlo a la creación de redes señuelo.

Las redes señuelo o, por brevedad *honeynets* a partir de ahora, son redes que están destinadas a que un usuario malicioso pueda atacar la red de una compañía sin causar impacto alguno en sus activos, mientras se le estudia su comportamiento dentro de la red. Estas redes suelen ser redes paralelas a la red de producción y, en mayor o menor medida, con grandes similitudes con ellas, ya que, una de las finalidades de las *honeynets* es que el posible atacante no advierta que está siendo estudiado.

Los estudios obtenidos monitorizando los pasos de este tipo de usuarios, una vez analizados de forma correcta y coherente, ofrecen una gran información acerca de los patrones utilizados para realizar estos ataques. Estos estudios pueden ser aplicados en la creación de reglas más específicas para Firewalls, IDS e IPS y con ello se reduce la posibilidad de un falso positivo a la hora de catalogar un tipo de conexión.

Una de las mayores desventajas de las *honeynets* es el gran esfuerzo monetario que se debe hacer para crear una *honeynet* cuya infraestructura sea proporcional a la infraestructura global de la red de producción, ya que, además de necesitar puestos terminales, son necesarios servidores y cualquier otro tipo de hardware que pueda ser esperado por estos atacantes.

Otra desventaja de las *honeynets* es la captura de los datos y el restablecimiento a un estado conocido de todos los sistemas de dicha *honeynet*, para estar preparada para el próximo ataque.

Estas desventajas son resueltas virtualizando los sistemas de la *honeynet* así como la propia red que los interconecta, haciendo que se necesiten menos recursos para crear el mismo entorno. Estos activos, aunque virtualizados, son reales de cara a un posible atacante por lo que no notará que está siendo estudiado mientras realiza los ataques dirigidos a la infraestructura.

Otra ventaja de la virtualización llega de la mano del análisis y del restablecimiento de la situación original del entorno. Simplemente, se debe realizar una copia del entorno virtualizado ya atacado a un lugar destinado para su análisis y otra copia de una imagen madre del entorno virtual a aquellos activos que estén implicados.

La creación de *honeynets* virtuales es compleja, ya que exige arrancar una serie de máquinas virtuales con distintos sistemas operativos e interconectarlas de acuerdo a una topología determinada. Surge por ello la necesidad de herramientas que ayuden y automaticen la gestión de los escenarios de red virtuales a usar en las *honeynets*.

En este sentido, el Departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid desarrolló la herramienta de gestión de escenarios virtuales *Virtual Networks over User Mode Linux (*VNUML), que ofrece la posibilidad de crear máquinas virtuales con sistema operativo Linux interconectadas mediante *switches* también virtuales.

Sin embargo, VNUML tiene varios inconvenientes. El primero es no poder realizar operaciones sobre una máquina virtual concreta, sino que se tiene que realizar sobre el escenario completo por lo que si es necesario reiniciar una máquina virtual en concreto, se debe reiniciar el escenario completo con la pérdida de datos que ello implica. Otro inconveniente es no poder virtualizar otros sistemas operativos excepto sistemas basados en Linux por lo que no responde a una red de producción real, ya que en estas existen típicamente un mayor número de máquinas con sistema operativo Windows que Linux.

Por todo ello se modificó VNUML para solventar estos inconvenientes y dar una imagen más realista al atacante de una red de trabajo normal. Con estas modificaciones se creó *Virtual Networks over linuX* (VNX), también desarrollado por el Departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid.

Una de las principales diferencias entre VNX y VNUML es la estructura modular, con la que se consigue la posibilidad de creación de otros módulos diferentes en los que se permita la virtualización de otros sistemas de una forma rápida y sencilla. Además, se consigue un mayor control sobre cada máquina virtual pudiendo realizar operaciones sobre estas sin que afecte al resto del entorno.

VNX soporta la virtualización mediante UML, la cual ha sido heredada del sistema anterior y adaptado para realizar una gestión individualizada de los sistemas. Además se ha añadido el soporte de libvirt, el API estándar de virtualización de Linux, el cual permite la virtualización de un mayor número de sistemas operativos como puede ser Windows, FreeBSD o cualquier distribución de Linux como *Suse* o *Debian*.

El desarrollo de VNX se ha enmarcado dentro del proyecto CENIT Segur@ (SEGUR@ - Seguridad y Confianza en la Sociedad de la Información). Este proyecto fue promovido por el Centro para el Desarrollo Tecnológico Industrial (CDTI), Organismo dependiente del Ministerio de Ciencia e Innovación. Se inició en Julio de 2007 dentro la tercera convocatoria del Programa CENIT (formando parte de la iniciativa INGENIO 2010) y terminó en el 2011.

Este proyecto SEGUR@, que contó con un presupuesto de más de 31 millones de euros, tuvo como objetivo generar un marco de confianza y seguridad para el uso de las TIC en la e-Sociedad en España. SEGUR@ tuvo una duración de más de 3 años y medio, y en él participaron 11 empresas y 15 universidades y centros de investigación, repartidos por 6 comunidades del Estado Español (1).

1.2 Estructura de la memoria.

La estructura de la memoria está diseñada para englobar y situar el proyecto dentro del ámbito que es la virtualización, para después pasar al desarrollo de las modificaciones que se introdujeron en la herramienta VNX para conseguir la virtualización de Windows y los routers Cisco.

Se empezará en el capítulo 2 con un análisis del estado del arte de las tecnologías de la virtualización que existen actualmente. Además en este capítulo se incluirá un análisis del software base que se utilizó en este proyecto, VNX y de su predecesor, VNUML.

En el capítulo 3 se detallará el desarrollo de la herramienta de VNXACE y su integración con Windows.

En el capítulo 4 se detallará el desarrollo que se ha procedido en la herramienta VNX para dar soporte a la virtualización de routers *Cisco* mediante *Dynamips*.

Una vez definido los capítulos principales en el capítulo 5 se expondrán las conclusiones a las cuales se ha llegado después de realizar el proyecto y se enumeraran los trabajos futuros que pueden realizarse a partir de lo expuesto en esta memoria.

Por último se ha incluido tres apéndices: El Apéndice A explica la instalación de la parte ejecutable en el equipo Windows virtualizado de la herramienta VNXACE; El Apéndice B, en el que se incluye un tutorial para el cálculo del parámetro idle_pc necesario para limitar el consumo de CPU cuando se virtualicen routers Cisco y el Apéndice C que contiene los listados de las funciones más importantes del proyecto.

Estado del arte

Spoon boy: Do not try and bend the spoon. That's impossible. Instead... only try to realize the truth.

Neo: What truth?

Spoon boy: There is no spoon.

Neo: There is no spoon?

Spoon boy: Then you'll see, that it is not the spoon that bends, it is only yourself.

Matrix. (1999)

2 Estado del arte

2.1 Introducción

En este capítulo, se pretende analizar el estado de la virtualización de hoy en día detallando los diferentes tipos de virtualización que existen. Con ello se pretende posicionar el proyecto dentro del rango de tecnologías existentes. Para completar el capítulo, se han introducido en la última parte los detalles de las soluciones de software de las que se han partido para realizar el proyecto.

En primer lugar, para dar una visión global se dará una definición de lo que significa virtualización: "La virtualización es la abstracción de los recursos de un ordenador, creando una capa entre la maquina física y el sistema operativo de la máquina virtual, haciendo que el sistema operativo de esta última puede ser arrancado como si estuviera instalado en una maquina física." (1)

Se pueden virtualizar un gran número de dispositivos que van desde videoconsolas antiguas hasta ordenadores personales completos pasando por routers y otros dispositivos de red. Todo este rango de dispositivos hace aún más interesante el uso de la virtualización, ya que puedes ser utilizado como método de desarrollo sin necesidad de adquisición del hardware al cual esté destinado, o la de validación de configuraciones de elementos de red antes de incluirla en la red de producción.

Hace unos años, la virtualización no se empleaba en entornos de producción y solo se utilizaba en grandes servidores con una gran capacidad de cálculo. Sin embargo, con el *boom* de internet, se fueron creando nuevos servicios y se necesitó más capacidad en estos servidores web de forma puntual por lo que se debería de hacer una gran inversión para dar respuesta a solo ciertos momentos del día. La virtualización fue la respuesta a estos problemas ya que permitía una gestión más inteligente de los recursos que se necesiten en ese momento y se daba solución a la creación de más servicios sin necesidad de adquisición de equipos informáticos. Además se añadían nuevas ventajas ya que la realización de copias de seguridad de estos servidores y la puesta en marcha de nuevo se realizaba de forma sencilla y rápida. Si el servicio se interrumpía, se podría levantar un nuevo servidor en cuestión de unos minutos.

Hoy en día es una de las soluciones más usadas por las compañías de diversos ámbitos ya que ofrece un ahorro de coste considerable. El 100% de las empresas que están en la

lista del TOP 100 Fortune Global utilizan la virtualización para su desarrollo de negocio (2). Adicionalmente, la virtualización está dentro de los que se llama Green Computing al ser una técnica que ayuda a preservar el medio ambiente. (3)

Por ultimo cabe destacar que numerosos estudios, así como documentos en revistas influyentes como Forrester citan la virtualización (4) como una de las medidas necesarias para el desarrollo óptimo del negocio.

A pesar de los numerosos informes a favor de la virtualización, no todos los tipos de virtualización ofrecen la misma respuesta y el mismo rendimiento. Por este motivo, es necesario elegir una tecnología acorde con el desarrollo que se va a realizar. La virtualización se puede clasificar en varios métodos dependiendo del nivel de hardware virtual se quiere hacer presentar al sistema virtualizado.

Estos métodos de virtualización pueden ser:

- Emulación
- Virtualización completa
- Paravirtualización
- Virtualización a nivel de sistema operativo
- Virtualización a nivel de aplicación.

En las siguientes secciones se detallará las principales características de cada sistema, incluyendo uno o más ejemplos de software que realiza una determinada virtualización.

2.2 Emulación.

La Emulación simula el hardware completo de un dispositivo por lo que cualquier sistema operativo que sea soportado por ese dispositivo puede ser ejecutado sin necesidad de modificaciones internas. Las ventajas de este modo es que no es necesario un procedimiento previo ni posterior de preparación del sistema operativo por la cual teóricamente puede ser compatible con cualquier sistema operativo. La principal desventaja es la gran cantidad de carga extra que debe soportar la máquina anfitrión para poder emular el hardware.

Uno de los sectores en que más se ha desarrollado la emulación ha sido en el sector de los videojuegos, ya que la posibilidad de poder ejecutar máquinas recreativas o incluso videojuegos creados para otras plataformas en ordenadores personales, hacían la emulación una práctica muy solicitada por toda la gente amante de los juegos antiguos.

Una de estas videoconsolas que fueron emuladas fue la GameBoy mediante el programa VisualBoyAdvance.

Fig. 1. VisualBoyAdvance ejecutando SuperMarioLand

Un emulador se divide en módulos que sigue el patrón del sistema emulado. Estos módulos son los siguientes:

- Emulador de la CPU.

- Emulador de memoria.
- Emuladores de I/O.

Una emulación sencilla de la CPU consistiría en seguir el flujo de ejecución traduciendo las órdenes correspondientes de una CPU, a otra sintaxis similar de la CPU destino donde se quiera ejecutar. Sin embargo, una desventaja importante es que el tiempo de ejecución se incrementa de forma considerable.

Un ejemplo de un software que realice este tipo de emulación es QEMU y Bochs.

Fig. 2. Sistema corriendo QEMU

2.3 Virtualización completa.

La virtualización completa permite la posibilidad de instalar un sistema operativo totalmente diferente al que está instalado en la máquina física. Sin embargo, a diferencia con el modo anterior, este modo no permite la conversión de plataforma, siendo necesario que el sistema operativo soporte la plataforma origen.

Para realizar esta labor, se instala una máquina virtual entre el sistema virtualizado y la maquina física llamado Hypervisor o VMM (Virtual Machine Monitor), como puede verse en la figura situada a la finalización de este párrafo. Dicho hipervisor ofrece la posibilidad de ejecutar el sistema operativo sin realizar cambio alguno en este.

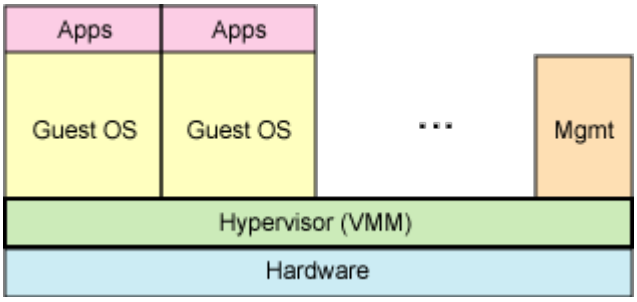


Fig. 3. Situación del Hypervisor (5)

El hipervisor es el encargado de realizar la traducción de las instrucciones de la máquina virtual a la máquina física para que puedan ser interpretadas por el procesador.

A lo largo del tiempo, la utilización de esta metodología de virtualización fue en aumento haciendo que incluso las máquinas de producción sean maquinas virtualizadas mediante programas como VMWare Workstation o Virtualbox. Esto es debido a su facilidad de uso, su eficiencia en cuanto a utilización de recursos y su escaso tiempo de reposición en caso de avería.

Este hecho ha producido que los dos grandes productores de CPU (Intel y AMD) hayan creado instrucciones específicas que permiten ejecutar determinadas instrucciones de la máquina virtual sin necesidad del paso por el Hypervisor, con esto se consigue aun mayor velocidad en la maquina virtualizada.

Como se ha comentado anteriormente, dos grandes ejemplos que usan este sistema de virtualización son VMWare y Virtualbox.

Ambos tienen un hipervisor como *core* del sistema, aunque su origen fue distinto. VMWare estuvo más orientado hacia el mundo empresarial mientras que Virtualbox

estuvo más orientado a dar una solución eficaz en Linux mediante una licencia GPL, aunque nació con una licencia privada.

Adicionalmente uno de los primeros en ejecutar una virtualización completa es el sistema de virtualización desarrollado por IBM en 1972 para sus sistemas z/OS llamado VM y que fue desarrollándose hasta llegar a la versión de hoy en día z/VM.

Un ejemplo de software que realice este tipo de virtualización son los programas anteriormente citados: VMWare Workstation, Virtualbox o z/VM.

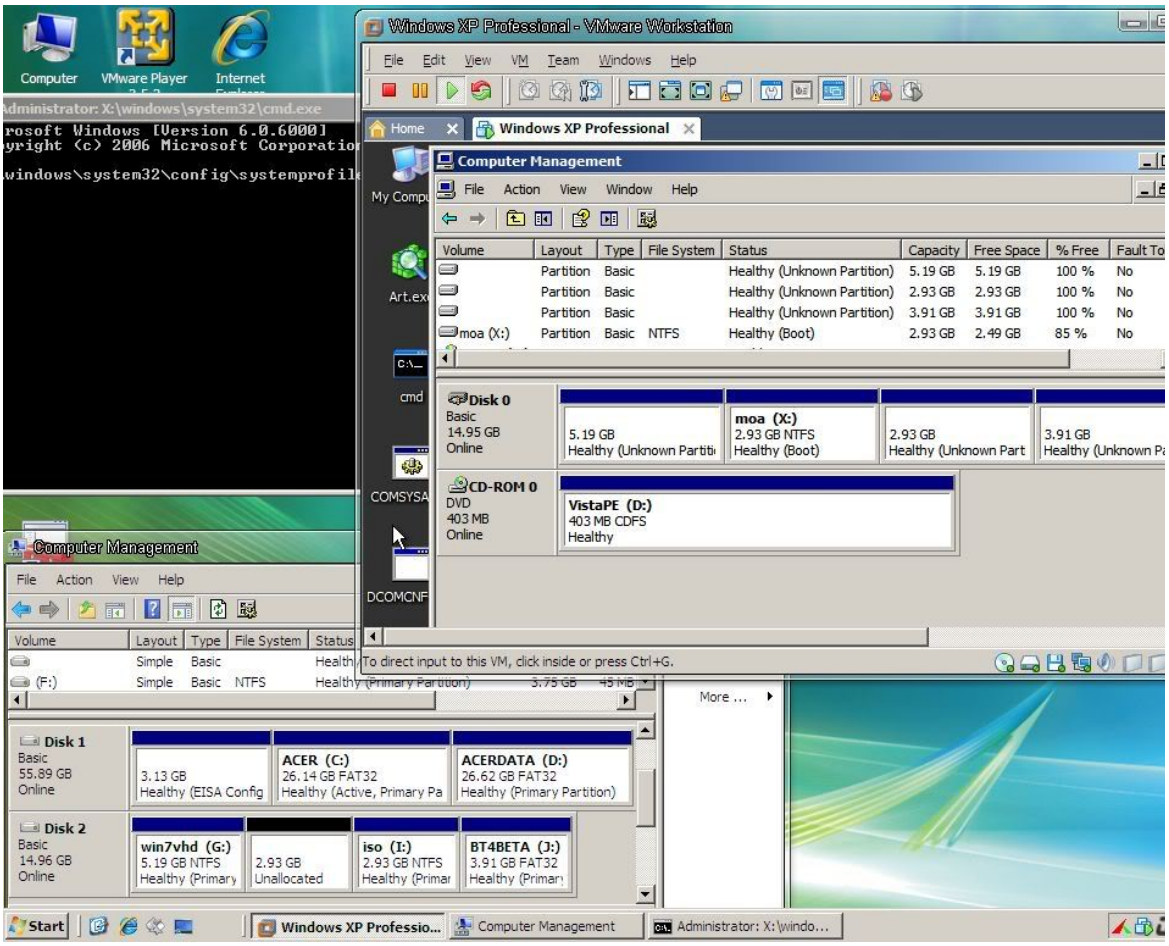


Fig. 4. Sistema ejecutando el software de virtualización VmWare

2.4 Paravirtualización

Al igual que los métodos anteriores, la paravirtualización permite la virtualización de los sistemas operativos mediante software. Para realizar su función, se aprovecha el hecho que los supervisores tienen varios niveles de ejecución, siendo el 0 el más privilegiado y el 3 el que menos privilegios tiene.

La paravirtualización utiliza estos niveles para dejar ejecutando el hipervisor en el nivel 0 mientras que el sistema operativo se ejecuta en el nivel 1. Este mecanismo permite un control sobre los recursos a los que se accede.

La paravirtualización no simula hardware alguno. Para virtualizar un sistema operativo en modo paravirtualización es necesario un sistema operativo que sea compatible ya que para su funcionamiento, tiene que llamar a procedimientos que están definidos en la API del sistema anfitrión. Actualmente cualquier sistema Linux permite ser ejecutado mediante paravirtualización. Sin embargo, los sistemas propietarios como Windows, no lo permiten, dado que sería necesaria la modificación de su núcleo para que pudiera ser paravirtualizado.

Con los nuevos procesadores, en los que se incluyen numerosos avances en la virtualización, se permite que los sistemas operativos sean ejecutados en el nivel 0, dejando al hipervisor un nuevo nivel llamado root-mode, el cual controla los componentes superiores. El resto de componentes se ejecutarían en el modo non-root-mode.

Un ejemplo de software que realice este tipo de virtualización es Xen. Este software está desarrollado por la universidad de Cambridge en 2003 y solo está disponible para los sistemas operativos de Linux y Unix. Es un sistema de alto rendimiento pudiendo llegar a un nivel de penalización de prestaciones de tan solo un 2%, lo que contrasta con los sistemas que se han visto anteriormente, que pueden llegar a tener una penalización de un 20%. (6)

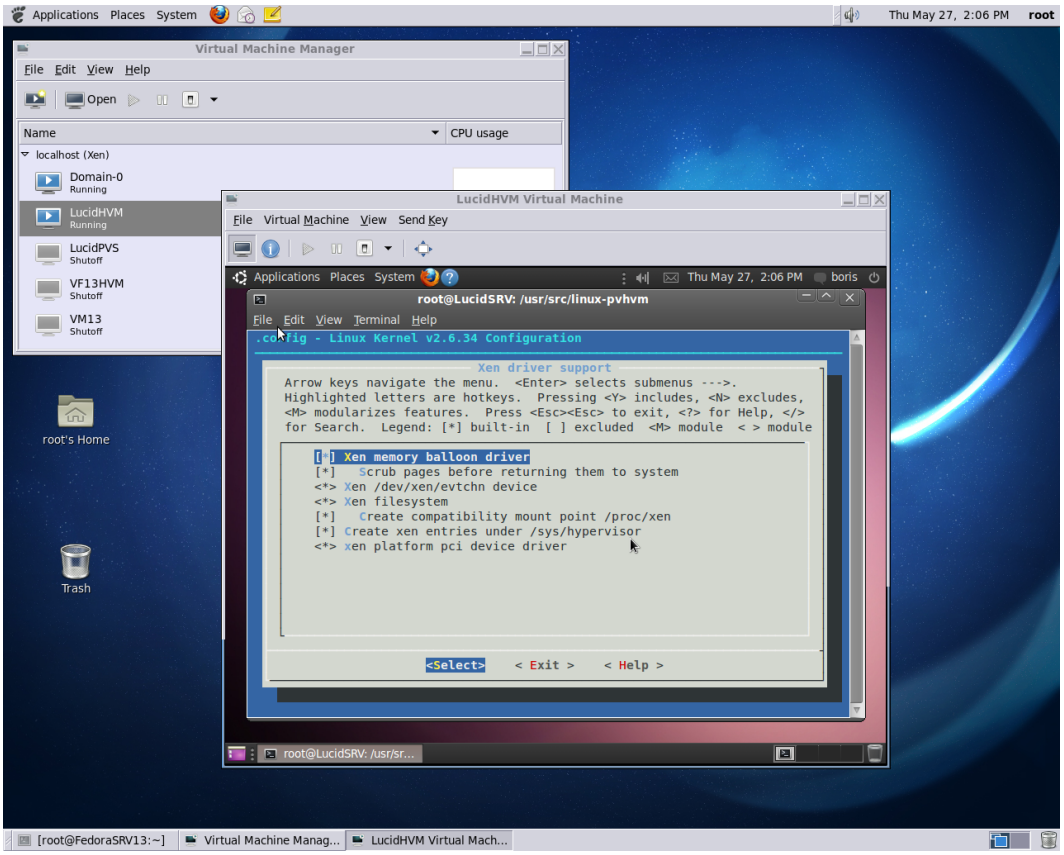


Fig. 5. Ejecución de Xen en un sistema Fedora.

En la actualizad, Xen pertenece a Citrix por lo que dicho software ha pasado a ser comercial, aunque sigue habiendo un producto gratuito llamado Citrix XenServer Free Edition con el inconveniente de que solo puede soportar cuatro máquinas virtuales.

2.5 Virtualización a nivel de sistema operativo.

Este modelo a diferencia de los anteriores, no virtualiza hardware alguno, sino crea copias del sistema operativo anfitrión para ponerlas a disposición de las aplicaciones. De esta manera, una aplicación que se estuviera ejecutando tendría la sensación de estar ejecutándose sobre un sistema operativo real sobre una máquina física.

En la siguiente figura se puede observar la estructura que utiliza el software de virtualización Parallels para realizar su virtualización.

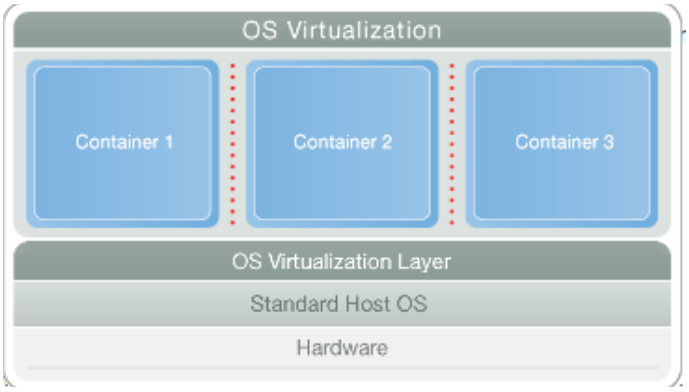


Fig. 6. Estructura de una virtualización a nivel de sistema operativo (7)

Dicho software mejora la gestión, el rendimiento y la eficiencia de los recursos definiendo varias capas. En primer lugar nos encontramos con el sistema operativo base, que en este caso puede ser un Linux y Windows. A continuación nos encontramos con el sistema que introduce Parallels para su virtualización. Esta capa creara de forma lógica varias celdas en las que se irán implementando y ejecutando las aplicaciones que se quieren introducir. Estas aplicaciones creerán que están en un servidor autónomo.

Al no necesitar de conversión de código ni traslación de comandos, este método obtiene un rendimiento mayor que las soluciones anteriores pero con el inconveniente de que su complejidad se agranda no permitiendo que sean varios sistemas operativos los que sean ejecutados en la misma máquina.

Un ejemplo adicional de software que permita hacer este tipo de virtualización seria mediante chroot disponible en los entornos UNIX y de uso gratuito a diferencia de Parallels como se ha comentado anteriormente.

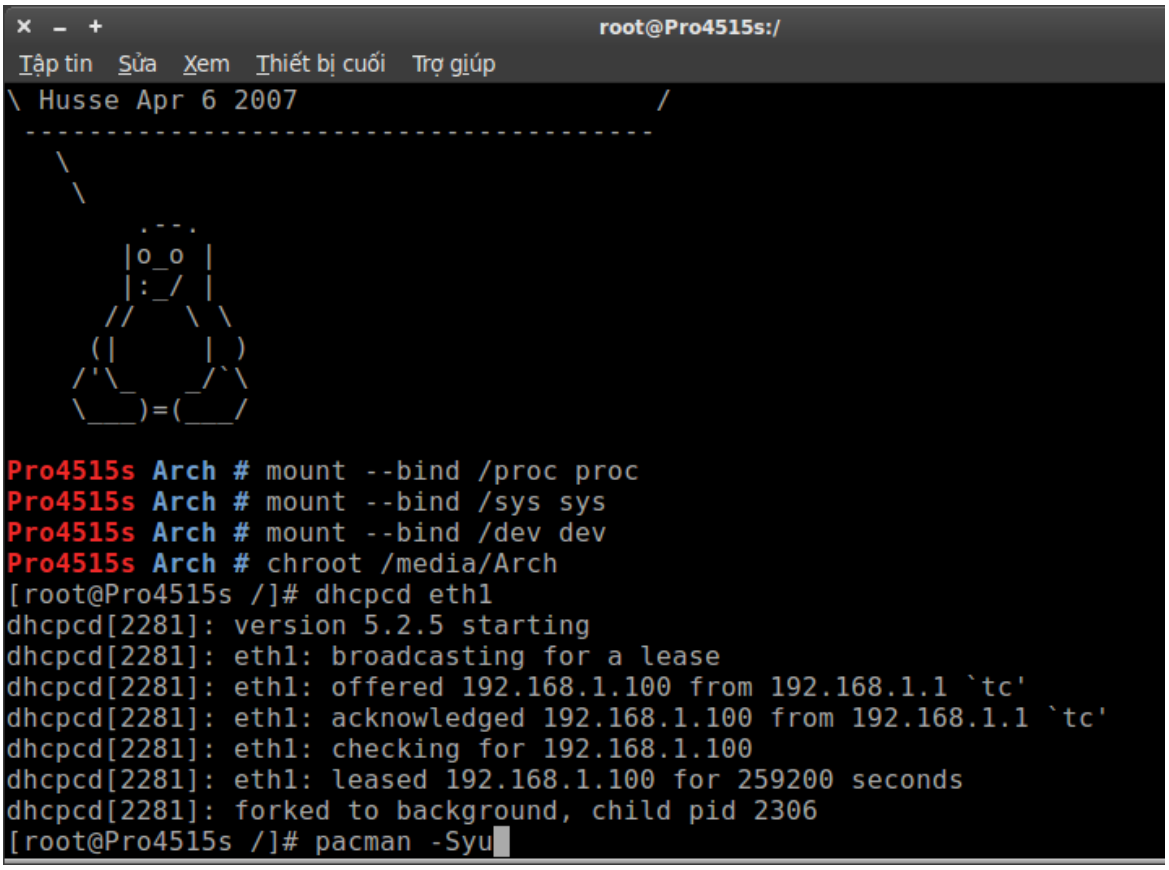


Fig. 7. Sistema ejecutando el sistema de virtualización chroot

Mediante este comando, se crea una pseudojaula en el que se controla los procesos de salida, entrada y aplicaciones que se ejecutan dentro de esa jaula, no permitiendo que las aplicaciones interaccionen de forma alguna con otras jaulas o con el sistema operativo que se ejecuta de forma global.

Las principales utilidades por las cuales se utiliza el chroot son las siguientes:

- Prueba y desarrollo
- Compatibilidad con software antiguo.
- Copias de seguridad
- Separación de privilegios

2.6 Virtualización a nivel de aplicación.

La virtualización a nivel de aplicación, al contrario que los ejemplos anteriores, no virtualiza ni hardware, ni sistemas operativos.

Este sistema utiliza un middleware que puede ser instalado en diferentes sistemas operativos y que proporciona una API común que pueden ser usados por los programas haciendo que el sistema operativo sea invisible a este.

Un ejemplo es el Java Virtual Machine de Oracle.

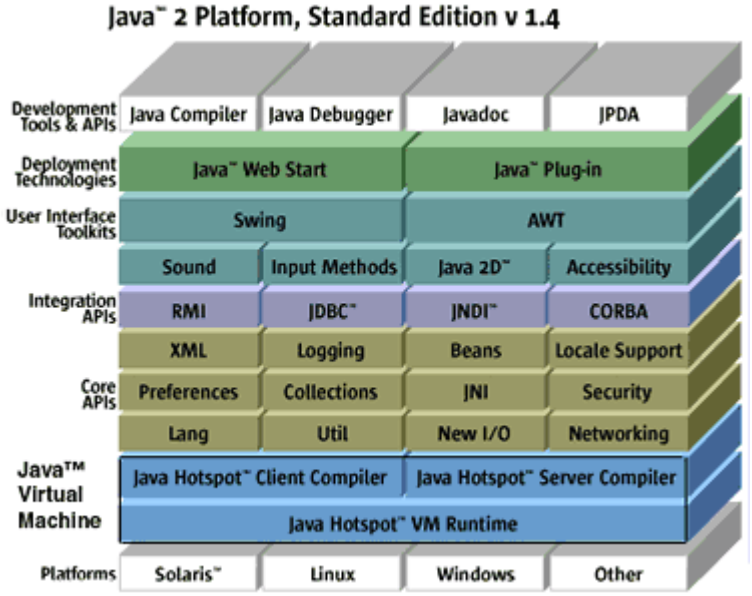


Fig. 8. Torre de protocolos del sistema de virtualización de Java

La ventaja de este sistema es la posibilidad de ejecución de la misma aplicación en diferentes sistemas operativos simplemente instalando este middleware que te crea en si una especie de máquina virtual.

El middleware básicamente provee de una API a las aplicaciones por la cual se permite realizar determinadas acciones. Una vez que la aplicación ha solicitado una determinada acción, el middleware la procesa y la convierte en determinadas llamadas al sistema operativo para realizar la acción solicitada por la aplicación.

2.7 OVF

OVF es la abreviación de Open Virtualization Format y es un estándar abierto de distribución de aplicaciones virtuales para que sean ejecutadas en máquinas virtuales. Este estándar fue liberado por el DMTF en Septiembre de 2008 a partir de los desarrollos que hicieron conjuntamente las compañías Dell, HP, IBM, Microsoft, VMWare y XenSource.

Las principales características de OVF se enumeran en la siguiente lista:

- OVF soporta la verificación y la integridad del contenido mediante el chequeo de clave pública, con ello provee un esquema básico de gestión de las licencias.
- OVF soporta la validación del paquete entero y la validación de cada componente del paquete OVF durante la fase de la instalación de la máquina virtual.
- Soporte para la instalación de diseños complejos en los que intervienen una o más máquinas virtuales en el proceso.
- OVF al ser estándar, no está diseñado específicamente para determinados sistemas de virtualización sino que el estándar describe un formato abierto, seguro, portable, eficiente y extensible para el empaquetado y distribución de software en máquinas virtuales.
- OVF es extensible pudiendo moldear el paquete para determinado recurso que se haya publicado con antelación.
- OVF es multidioma por lo un mismo paquete puede utilizarse en diversos idiomas.

Debido a que OVF es un estándar abierto, no debe fijar parámetros obligatorios que solo sean aceptados por un programa determinado, por eso, las imágenes de los discos virtuales no están definidas solo requiriendo que lo acepte el sistema final donde se va a implantar el sistema.

El paquete puede está formado por:

- Un fichero de descripción con extensión .ovf.
- Cero o un fichero tipo manifest con extensión .mf.
- Cero o un certificado OVF con extensión .crt.
- Cero o más imágenes de disco.

- Cero o más ficheros adicionales que puedan ser necesitados como ficheros ISO.

Este paquete se puede distribuir en un único fichero con extensión .ova (open virtual appliance) que es simplemente un fichero comprimido en formato TAR en el que contiene uno o más de los ficheros que se han indicado arriba del mismo. Adicionalmente también se puede distribuir fichero por fichero siendo esta opción ideal si se va a publicar vía web.

El fichero principal que describe la aplicación es el fichero con extensión .ovf. Este fichero está formado por un XML cuyo formato está definido por el fichero de esquema ovf-envelop.xsd.

Un ejemplo de un fichero OVF puede ser el siguiente.

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:vssd="http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/CIM_VirtualSystemSettingData"
xmlns:rasd="http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/CIM_ResourceAllocationSettingData"
xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
xmlns="http://schemas.dmtf.org/ovf/envelope/1"
xml:lang="en-US"> <References>
<File ovf:id="de-DE-resources.xml" ovf:size="15240"
ovf:href="http://mywebsite/virtualappliances/de-DE-resources.xml"/>
<File ovf:id="file1" ovf:href="vmdisk1.vmdk" ovf:size="180114671"/>
<File ovf:id="file2" ovf:href="vmdisk2.vmdk" ovf:size="4882023564"
ovf:chunkSize="2147483648"/>
<File ovf:id="file3" ovf:href="resource.iso" ovf:size="212148764"
ovf:compression="gzip"/>
<File ovf:id="icon" ovf:href="icon.png" ovf:size="1360"/>
</References> <!-- Describes meta-information about all virtual disks in the
package -->
<DiskSection>
<Info>Describes the set of virtual disks</Info> <!-- Additional section
content -->
</DiskSection>
<!-- Describes all networks used in the package -->
<NetworkSection>
<Info>List of logical networks used in the package</Info> <!-- Additional
section content -->
</NetworkSection>
<SomeSection ovf:required="false">
<Info>A plain-text description of the content</Info>
<!-- Additional section content -->
</SomeSection>
<!-- Additional sections can follow -->
<VirtualSystemCollection ovf:id="Some Product">
<!-- Additional sections including VirtualSystem or VirtualSystemCollection-->
</VirtualSystemCollection >
<Strings xml:lang="de-DE">
<!-- Specification of message resource bundles for de-DE locale -->
</Strings>
</Envelope>
```

Listado 1. Ejemplo fichero OVF

2.8 Dynamips

Dynamips es un emulador de routers *Cisco*® desarrollado por Christophe Fillot que es capaz de emular los modelos 1700, 2600, 3600, 3700 y 7200 de Cisco con diverso hardware asociado.

Según su creador, este emulador fue desarrollado para:

- Familiarizarse con los routers de la marca *Cisco*.
- Experimentar con las diversas funcionalidades de los routers.
- Probar configuraciones que se implantarán después para saber su impacto en la red.

Adicionalmente en el paquete de *Dynamips*, se instala su gestor *Dynagen*. Este gestor es un "front-end" de *Dynamips* que actúa como hipervisor. Otro gestor que realiza las mismas funciones de *Dynagen* con una interfaz más amigable es GNS3, el cual te permite crear gráficamente la red virtual con routers, *firewalls* y host virtualizado. Ambas se comunican con *Dynamips* mediante comandos de texto que se envían a un puerto concreto en el que *Dynamips* escucha.

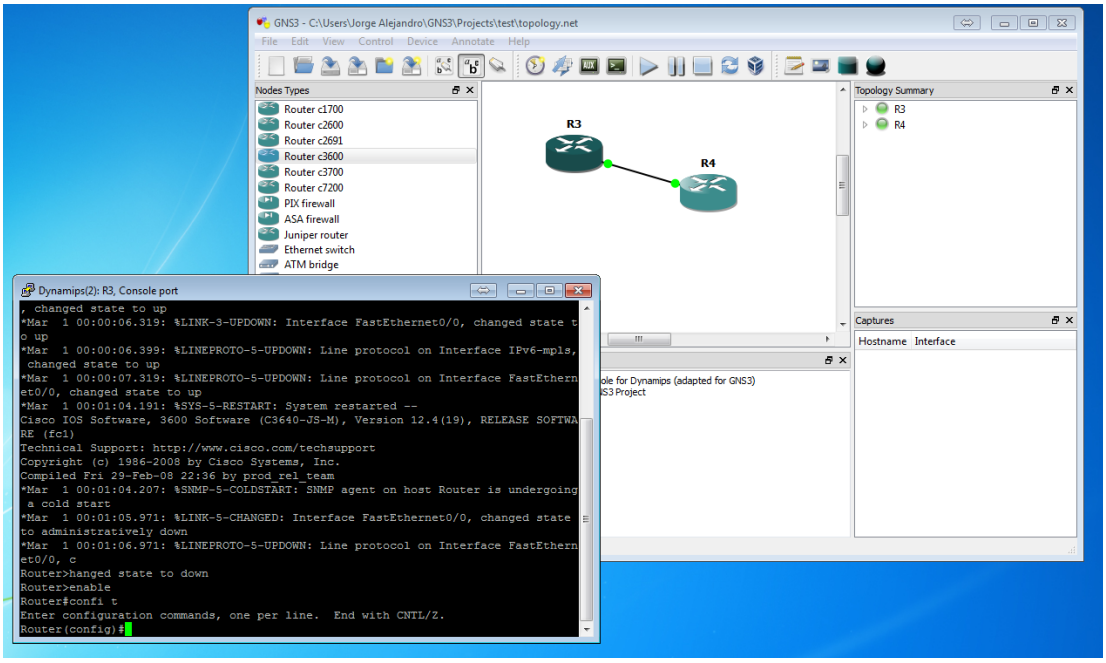


Fig. 9. Software GNS3 ejecutando un escenario virtual con routers Cisco

Estos gestores están pensados para ayudar en la configuración de los escenarios mediante sencillos comandos que son traducidos a posteriori en uno o varios comandos adecuados al lenguaje que utiliza *Dynamips* interiormente.

En la siguiente captura se muestra una consola *Dynagen* que está conectada a un hipervisor *Dynamips* en el que están ejecutándose dos routers *Cisco*.

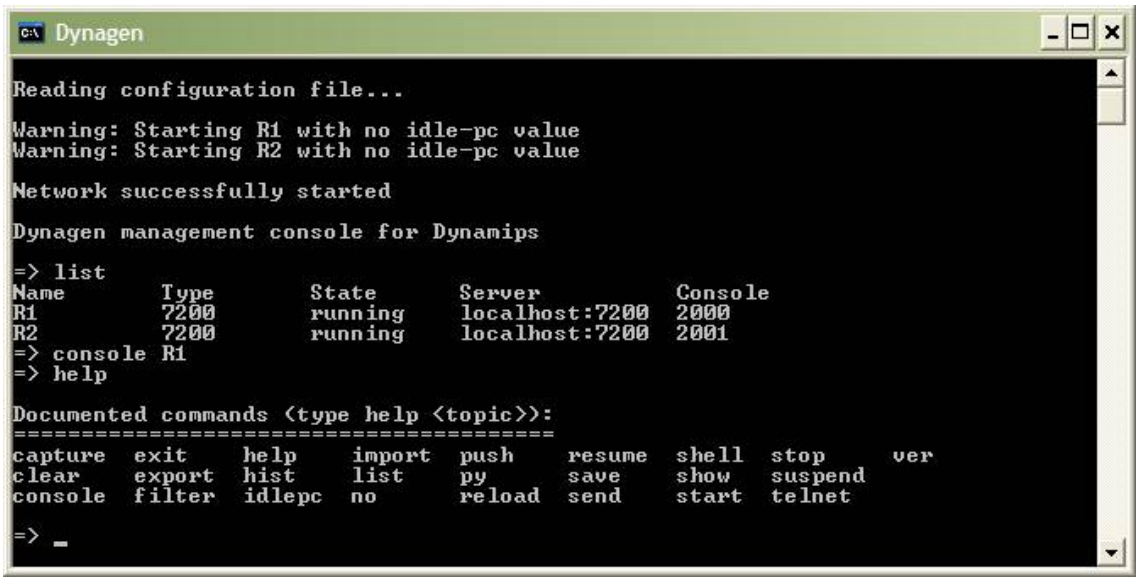


Fig. 10. Ejecución de *Dynamips*

Para poder ejecutar *Dynamips* es necesario tener instalado una imagen oficial del sistema operativo que desarrolla *Cisco* para sus dispositivos llamado CiscoIOS. Es por ello por lo que el autor del proyecto detuvo su evolución en octubre de 2007 siendo la última versión la 0.2.8-RC2 que es la que actualmente se puede encontrar en los repositorios de las principales distribuciones de Linux como Ubuntu o Fedora.

Al estar parado el proyecto, su documentación sobre sus comandos es mínima necesitando realizar un proceso de ingeniería inversa para obtener los comandos soportados a través de los sistemas de gestión *Dynagen* y *GNS3* nombrados antes. Este procedimiento se explicará con más detalle en el apartado dedicado a la Integración de *Dynamips*. Aunque en este apartado se dará un detalle mayor sobre que funciones son las primordiales y en qué orden se deben ejecutar se presenta a continuación una tabla en la que se resumen los comandos principales así como una descripción de su funcionalidad.

Comando	Interpretación
`hypervisor version`	Resetea *dynamips* parando todos los routers que pudieran estar en ejecución
`hypervisor working_dir <directorio>`	Establece que todos los ficheros, directorio temporales,… sean debajo del directorio indicado
`vm create <nombre de la máquina virtual> 0 <modelo>`	
`vm set_con_tcp_port <nombre> <puerto consola>`	Define el puerto donde va a residir la consola principal del Cisco
`c<modelo> set_chassis <nombre> <submodelo>`	Define el chasis que tendrá el modelo en concreto
`c<modelo> set_npe <nombre> npe-<valor>`	Establece el valor del NPE solo disponible en routers cisco de la familia 7200
`vm set_ios <nombre> <ruta del img>`	Establece la ruta donde se encuentra la imagen oficial del router cisco
`vm set_ram <nombre> <cantidad de memoria>`	Establece la cantidad de memoria RAM que se va a reservar para dicho router
`vm set_sparse_mem <nombre> 1`	1 para usar la opción de *sparse memory* o 0 para no usar *sparse memory*
`vm set_idle_pc <nombre> <valor del idle_pc>`	Establece el valor de idle_pc, esto es necesario para que el router no esté usando todo el procesador cuando no está trabajando.
`vm set_ghost_status <nombre> 2`	Establece la opción de IOS Ghost
`vm set_ghost_file <nombre> <ruta>`	Si la opción del IOS Ghost está activada, se indica la ruta.
`vm set_blk_direct_jump <nombre> 0`	Establece si se utiliza la opción set_blk_direct_jump
`vm slot_add_binding <nombre> <indice> 0 <slot>`	Crea un slot para la introducción de una tarjeta.
`nio create_tap nio_tap_<nombre><slot><dev> <nombreTAP>`	Crea una interfaz de red.

Comando	Interpretación
`vm slot_add_nio_binding <nombre> <slot> <device> nio_tap_<nombre><slot><dev>`	Añade una tarjeta de red a la interfaz creada anteriormente.
`nio create_udp nio_udp_<nombre><slot><dev> <Puerto1> 127.0.0.1 <puerto2>`	
`vm slot_add_nio_binding <nombre> <slot> <dev> nio_udp_<nombre><slot><dev>`	
`vm set_config <nombre> <ruta al fichero de configuración>`	Establece el fichero de configuración en el router.
`vm stop <nombre>`	Detiene el router virtual
`vm delete <nombre>`	Elimina el router virtual del sistema de *Dynamips*
`vm start <nombre>`	Arranca el router virtual
`vm extract_config <nombre>`	Extrae la configuración que posee actualmente el router en el sistema
`vm suspend <nombre>`	Suspende el router virtual en *Dynamips*
`vm resume <nombre>`	Vuelve de la suspensión el router virtual

2.9 VNUML

VNUML (Virtual Network User-Mode-Linux) es una herramienta de simulación que permite lanzar escenarios de varios sistemas con kernel Linux que están interconectados entre ellos de una forma que el propio usuario puede elegir qué tipo de escenario necesita solo con editar unos ficheros XML. Esta herramienta fue desarrollada por el Departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid.

Mediante esta herramienta se permite crear redes personalizadas para recrear cualquier tipo de tráfico y topología para poder probar como se comportarán aplicaciones de red a diferentes tipos de eventos que se puedan producir. Otro gran uso que se le puede dar a la herramienta es la creación y ejecución de máquinas virtuales con distintas configuraciones para crear *honeynets* "al vuelo".

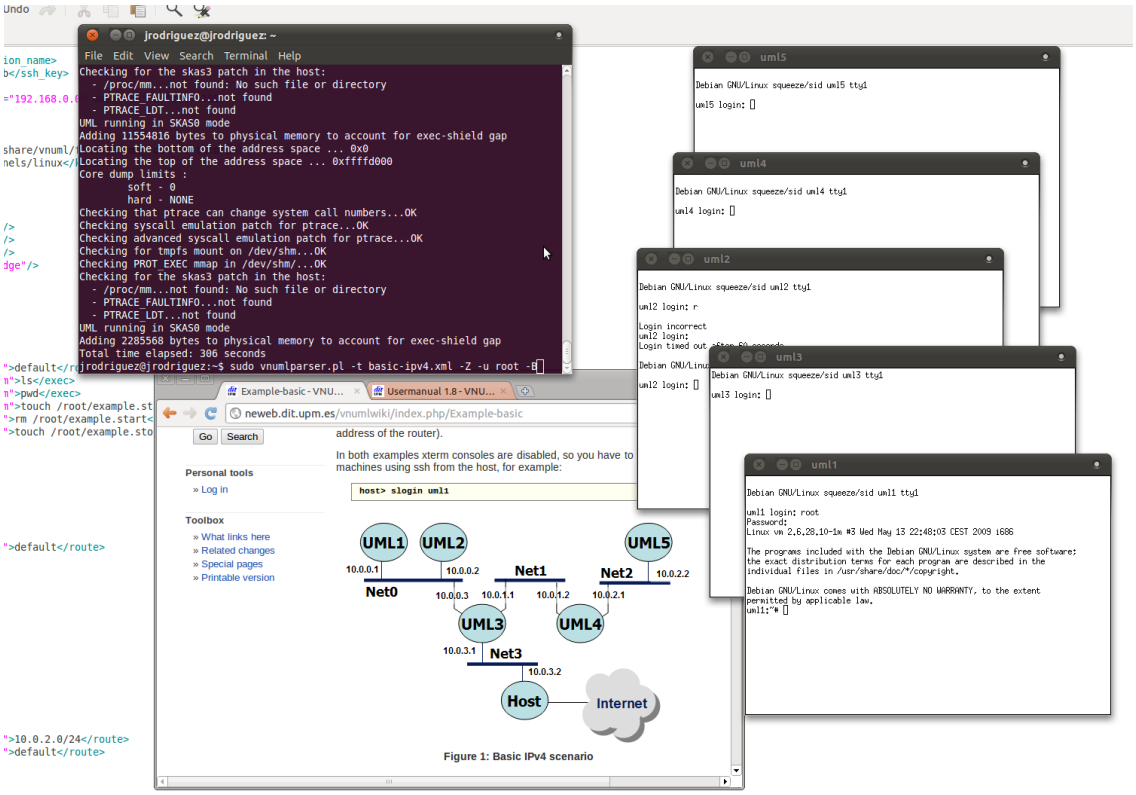


Fig. 11. Ejemplo de un escenario creado mediante VNUML

VNUML tiene una carencia que viene impuesta por la utilización de UML (User-Mode-Linux) (8) como base y es que no permite emular maquinas con un kernel diferente a Linux. En ciertos entornos como, por ejemplo, las *honeynets*, es necesario crear redes lo más parecidas a las redes de producción reales que, por lo general, están compuestas

por estaciones de trabajo con Windows® como sistema operativo, entre otros. Además, estas estaciones de trabajo normalmente también se diferencian entre sí por el software, actualizaciones y parches que están instalados, que son los principales actores encargados de tener un mayor o menor impacto ante un posible ataque. (8)

Para solucionar esta carencia, se ha modificado VNUML para dar soporte a la virtualización de un mayor rango de sistemas operativos y no limitarse exclusivamente a UML.

Esta modificación ha dado lugar a la creación de una nueva herramienta llamada VNX, (descrita en la siguiente sección), que permite salvar los anteriores inconvenientes mediante el uso del API estándar de virtualización de Linux libvirt u otro sistema de virtualización que se desarrolle y se adapte a la modulación de VNX. Además, añade además otras funcionalidades nuevas necesarias para la consecución de los objetivos marcados. A pesar de los cambios realizados, se ha mantenido la compatibilidad para que VNX pueda ejecutar máquinas creadas para VNUML solo configurando adecuadamente el fichero de definición del entorno.

2.10 VNX

El objetivo básico que guió el desarrollo de la nueva herramienta denominada Virtual Networks over linuX (VNX) fue la generalización de VNUML para poder virtualizar otros sistemas operativos diferentes a Linux, mediante el uso de otros hipervisores diferentes a UML. Dada la disponibilidad en Linux del API estándar libvirt (9) para el acceso a los diversos entornos de virtualización disponibles, se decidió diseñar y desarrollar VNX alrededor de este estándar.

Para realizar el objetivo básico, es necesario abordar, entre otros, los siguientes puntos:

- Desarrollar e implementar un método que permita la autoconfiguración de máquinas virtuales de forma genérica y segura. Para ello, se plantea la definición de un servicio en el sistema virtualizado que realice la autoconfiguración a partir de los datos incluidos en un fichero XML que se reciba desde el sistema anfitrión. El mecanismo elegido para proporcionar los datos desde el anfitrión a la máquina virtual es la inserción de discos CDROM emulados, tal como se propone en los estándares OVF Environment.
- Desarrollar e implementar un método que permita la ejecución de comandos en el sistema virtualizado. Como en el punto anterior, también se ha planteado la definición de un servicio que ejecute los comandos que se le vaya insertado en discos en caliente.

El servicio anterior como mínimo se tendrá que desarrollar para Windows, Linux y FreeBSD, que son los principales sistemas operativos que actualmente se puede encontrar en una red de producción. Sin embargo, en este proyecto se ha desarrollado el sistema exclusivamente para entornos Windows, en paralelo con el desarrollo para el resto de plataformas las cuales fueron realizadas por otros miembros del grupo de desarrollo.

Como se ha comentado anteriormente, para la realización de VNX se tomó como referencia el software de virtualización VNUML, desdoblando su estructura para permitir su expansión de forma de módulos y la gestión individualizada de las máquinas virtuales.

VNX permite la creación de escenarios virtuales sobre un único ordenador, este escenario está compuesto por redes virtuales creadas con los propios switches

virtualizados que pone a disposición libvirt y sistemas operativos virtualizados gestionados por él, que están conectados a estos switches virtuales.

Una de las desventajas que supone VNX es la casi imposibilidad de comunicación que posee con el sistema operativo virtualizado por lo que es necesario un desarrollo posterior de alguna herramienta con la que se pueda tener una comunicación entre el equipo virtualizado y VNX. Adicionalmente, por utilizar la tecnología de libvirt como capa inferior, esta no puede virtualizar routers por lo que se utilizará *Dynamips* para estaba en desarrollo.

VNX es la base de este proyecto, ya que el desarrollo realizado corresponde a suplir las dos desventajas mostradas anteriormente. También se ha de comentar que la realización se ha llevado en paralelo con la maduración de VNX ya que se ha tenido la posibilidad de realizar una colaboración para el desarrollo de VNX mientras el proyecto de VNXACE tomaba forma.

Una captura de pantalla de la herramienta ejecutando un entorno virtual es la siguiente:

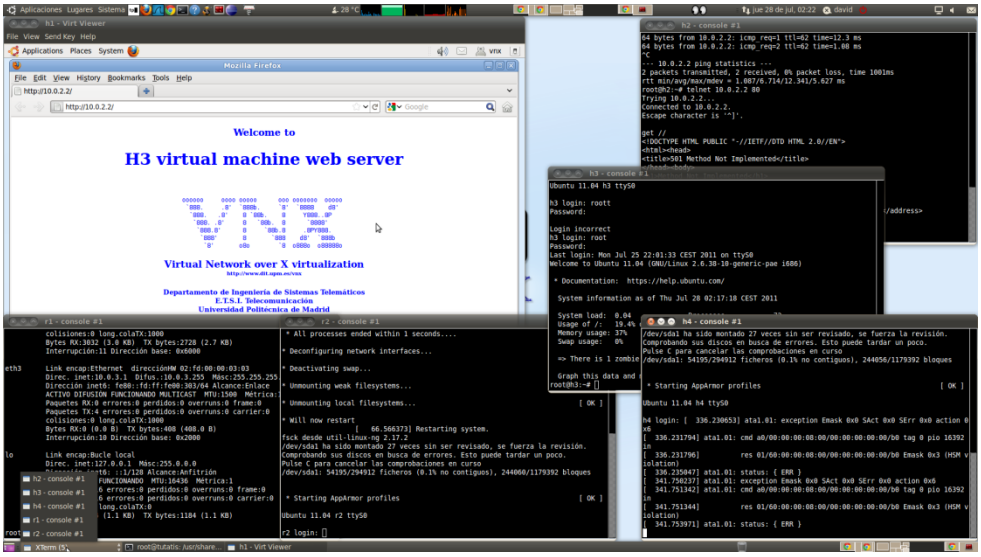


Fig. 12. Captura de pantalla de la herramienta VNX

VNX fue concebido para que su ejecución fuera lo más modular posible, separándolo en dos grandes áreas que se comunican mediante una API. Esto se realizó de esta determinada forma para poder dar una mayor facilidad de creación de módulos para otros sistemas operativos.

En la siguiente figura se muestra los componentes que se utilizan en una virtualización donde están involucrados el sistema UML, Libvirt y *Dynamips*.

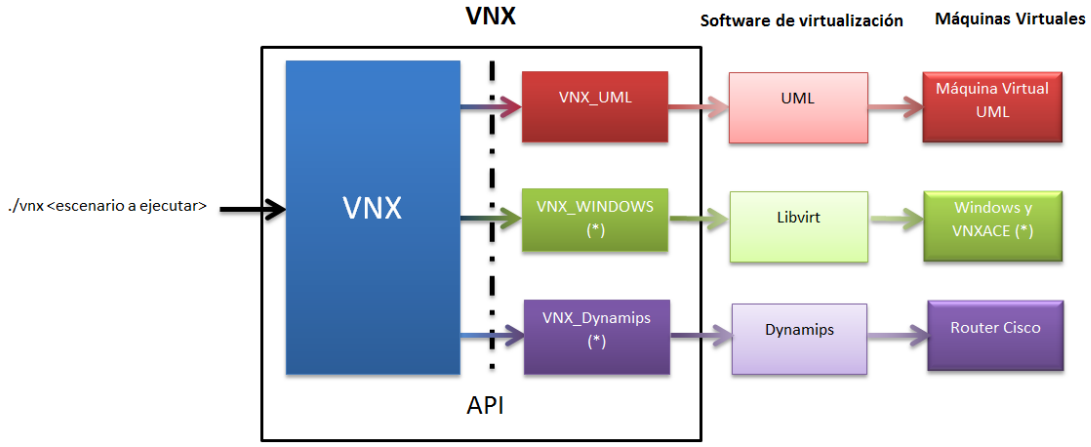


Fig. 13. Arquitectura general de VNX

Para situarse de una forma más puntual, se ha procedido a marcar con el símbolo (*) aquellos componentes de la figura que se han desarrollado en este proyecto.

Estas dos partes corresponden a una común, en el que está la lógica de creación de los ficheros temporales y el manejo de todas las máquinas virtuales que están en ejecución en un determinado instante, y la otra parte corresponde a un desarrollo individualizado por cada sistema operativo o sistema de virtualización.

Estas dos partes se comunican mediante un API que se expone a continuación:

Nombre de función	Función
init	Inicialización de la función
defineVM	Definición de la máquina virtual en el hipervisor y todos sus ficheros asociados pero sin arranque en la máquina virtual.
undefineVM	Eliminación de la máquina virtual del hipervisor y de todos sus ficheros asociados.
destroyVM	Parada y eliminación de todos los ficheros asociados a la máquina virtual
startVM	Arranque de la máquina virtual en el hipervisor y lanzamiento de las consolas para su administración si así procede.
shutdownVM	Apagado ordenado de la máquina virtual en el hipervisor pero sin la eliminación de sus ficheros asociados.

Nombre de función	Función
saveVM	Salva del estado actual de la máquina virtual dentro del hipervisor.
restoreVM	Recuperación de una máquina virtual que anteriormente estaba suspendida.
suspendVM	Suspensión de una máquina virtual en el hipervisor.
resumeVM	Recuperación de una máquina virtual que anteriormente estaba suspendida.
rebootVM	Ordena re arranque ordenado de la máquina virtual al hipervisor.
resetVM	Ordena el re arranque inmediato de la máquina virtual.
executeCMD	Ordena la ejecución de un determinado comando

Para realizar un nuevo módulo, simplemente deben implementar las funciones que responden a las llamadas anteriores para que hagan la función a las que están diseñadas. Estas llamadas se traducirán en otras llamadas a funciones del nuevo gestor. Adicionalmente, no se necesita tomar en cuenta posibles concurrencias con otros sistemas de virtualización ya que esa parte se gestiona directamente a través de VNX.

Una de las finalidades de esta herramienta es la creación automática de *honeynets*. Esto junto con herramientas que tracen el comportamiento de los posibles atacantes como pueda ser Sebek (10), hacen que el sistema completo se convierta en una herramienta muy interesante en la localización, aprendizaje y defensa de posibles ataques a redes completas.

Autoconfiguración y ejecución de comandos en Windows

Ariadne: Why are they all looking at me?

Cobb: Because my subconscious feels that someone else is creating this world. The more you change things, the quicker the projections start to converge on you.

Ariadne: Converge?

Cobb: It's the foreign nature of the dreamer. They attack like white blood cells fighting an infection.

Ariadne: They're going to attack us?

Cobb: No. Just you.

Inception (2010)

3 Autoconfiguración y ejecución de comandos en Windows.

3.1 Introducción

En este capítulo se procederá a detallar el desarrollo que se ha realizado del sistema de autoconfiguración y ejecución de comandos (VNXACE) en C++, así como el proceso de integración dentro de la herramienta VNX para dar soporte a la virtualización de sistemas Windows.

KVM no permite la configuración completa de parámetros internos de la máquina virtual, por lo que se hacía necesario desarrollar algún sistema que permitiera proceder a configurar dichos parámetros desde el host.

Otra de las necesidades que se planteó inicialmente fue que el método utilizado para transferir esos parámetros de la maquina host a la máquina virtual no pudiera comprometer la maquina host en caso que la máquina virtual fuera infectada por algún tipo de software malicioso.

Además, se necesitaba la posibilidad de ejecutar comandos dentro de la máquina virtual de software que estuviera ya preinstalado como de software nuevo sin necesidad de parar la máquina virtual para modificarla.

Con estas dos necesidades, se planteó la solución de desarrollar un programa propio instalable en la máquina virtual el cual obtuviera los parámetros de configuración de la unidad lectora de CD y fuera configurando automáticamente la máquina virtual respecto a esos parámetros. Estos datos deben ser introducidos previamente en una imagen ISO autogenerada que contiene los parámetros de configuración de esa máquina virtual, esta imagen se asocia a la unidad lectora de CD de la máquina virtual previo al encendido de dicha máquina para proceder a la autoconfiguración.

Este programa también debe soportar la inserción de programas ajenos, copia de estos y su ejecución. Este método también se realiza mediante la inserción de una imagen ISO autogenerada que contiene el programa a copiar a la máquina virtual y un fichero que contiene los parámetros tales como la dirección destino a donde copiar los ficheros y la línea de ejecución final.

3.2 Creación de VNXACE

3.2.1 Descripción

Uno de los requisitos para el desarrollo de VNXACE es la compatibilidad con los sistemas operativos Windows XP y 7 al mismo tiempo sin necesidad de creación de un fichero por plataforma.

Estos sistemas operativos son muy diferentes entre sí dando lugar a incompatibilidades si se centra el desarrollo en uno de ellos. Una de las diferencias relevantes de estos sistemas operativos que supuso un reto para la creación de VNXACE fue la nueva implementación de medidas de seguridad específicas por parte de los sistemas operativos para los niveles de ejecución de los ficheros.

Estas medidas hacían que los procesos ejecutados con un determinado usuario no se le permitiesen acceder a determinados recursos del sistema operativo evitando así posibles ataques como pueden ser la elevación de privilegios no controlada o la ejecución arbitraria de código. Un ejemplo es que un proceso ejecutado mediante la cuenta de usuario SYSTEM no tiene acceso al sistema de eventos por lo que VnxDaemon no puede recibir los eventos de inserción de CD.

Por este motivo VNXACE se dividió en tres partes diferenciadas: VnxService, VnxDaemon y VnxClient. Cada una de las partes tiene una finalidad concreta que a continuación detallamos.

- VnxService.

  Esta parte se encarga de controlar la ejecución de VnxDaemon, lanzándolo en caso que no esté en ejecución. Se inicia con el usuario SYSTEM, el cual permite la configuración de todos los parámetros del sistema. Se instala como servicio de Windows para que se ejecute nada más ser cargada la parte de red de Windows.

- VnxDaemon.

  Este es el programa principal del sistema. Se encarga de realizar los cambios necesarios en la máquina virtual de acuerdo al fichero de configuración que se le ha compartido mediante CD-ROM. Estos cambios son posibles porque hereda el usuario con el que se ha ejecutado, que en este caso es SYSTEM. Una vez

realizada la configuración de los parámetros se queda a la espera de recibir los avisos de VnxClient para recibir por parte de este la unidad donde se ha insertado el nuevo CD virtualizado. Esto es debido a que este usuario no posee los permisos necesarios para obtener determinadas señales como puede ser la inserción de un CD en la unidad lectora por lo que se hace indispensable una tercera parte ejecutándose desde otro usuario para que se comunique con VnxDaemon ante el evento de inserción de CD.

- VnxClient.

  Esta parte se encarga de la notificación de inserción de dispositivos a VnxDaemon. Esta parte es ejecutada automáticamente una vez que un usuario es autenticado en el sistema. Al ser ejecutado con un usuario nominal y no de sistema, el VnxClient tiene acceso a los eventos de inserción pudiendo obtenerlos simplemente registrándose en el manejador de eventos de Windows.

  Adicionalmente VnxClient posee una parte gráfica la cual informa de los eventos producidos por el sistema y por VnxDaemon a tener una comunicación bidireccional con este.

Fig. 14. Descripción gráfica del sistema

3.2.2 Orden de ejecución.

En primer lugar, una parte del complejo proceso de inicio del sistema Windows, es el inicio de los servicios que tiene instalados, entre ellos el VnxService. Al ser este servicio el controlador de que VnxDaemon este continuamente ejecutando, este lo inicia automáticamente para que quede en segundo plano.

Una vez que VnxDaemon se ha iniciado, procederá a la autoconfiguración del sistema operativo tal y como estén en las especificaciones que se le habrán compartido a través del fichero VNXBOOT existente en el lector de CD de la máquina virtual del que se habrá adjuntado anteriormente al encendido de la máquina virtual. Los cambios producidos por la autoconfiguración del sistema contienen cambios en el sistema de red y el nombre de la máquina virtual. Si el nombre no es el adecuado respecto a lo expuesto en el fichero de autoconfiguración, VnxDaemon cambia el nombre de la máquina y la reinicia para que los cambios queden presentes. Una vez realizada la autoconfiguración, VnxDaemon queda a la espera de conexión por parte del cliente VnxClient para recibir la letra de unidad de CD en el que se le haya insertado un nuevo disco.

VnxClient es iniciado de forma independiente al resto de las partes cuando algún usuario es autenticado en el sistema. Una vez que se ha iniciado, queda a la espera de recibir los eventos de inserción de CD para reenviarle la letra de la unidad lectora a VnxDaemon para que este lo procese.

Un flujograma del funcionamiento de la aplicación se puede observar en la siguiente página, enumerado con el identificado Fig 15.

En los siguientes capítulos se detallarán el flujo que realizan las partes de VnxDaemon y VnxClient detallando las funciones más relevantes para el proyecto.

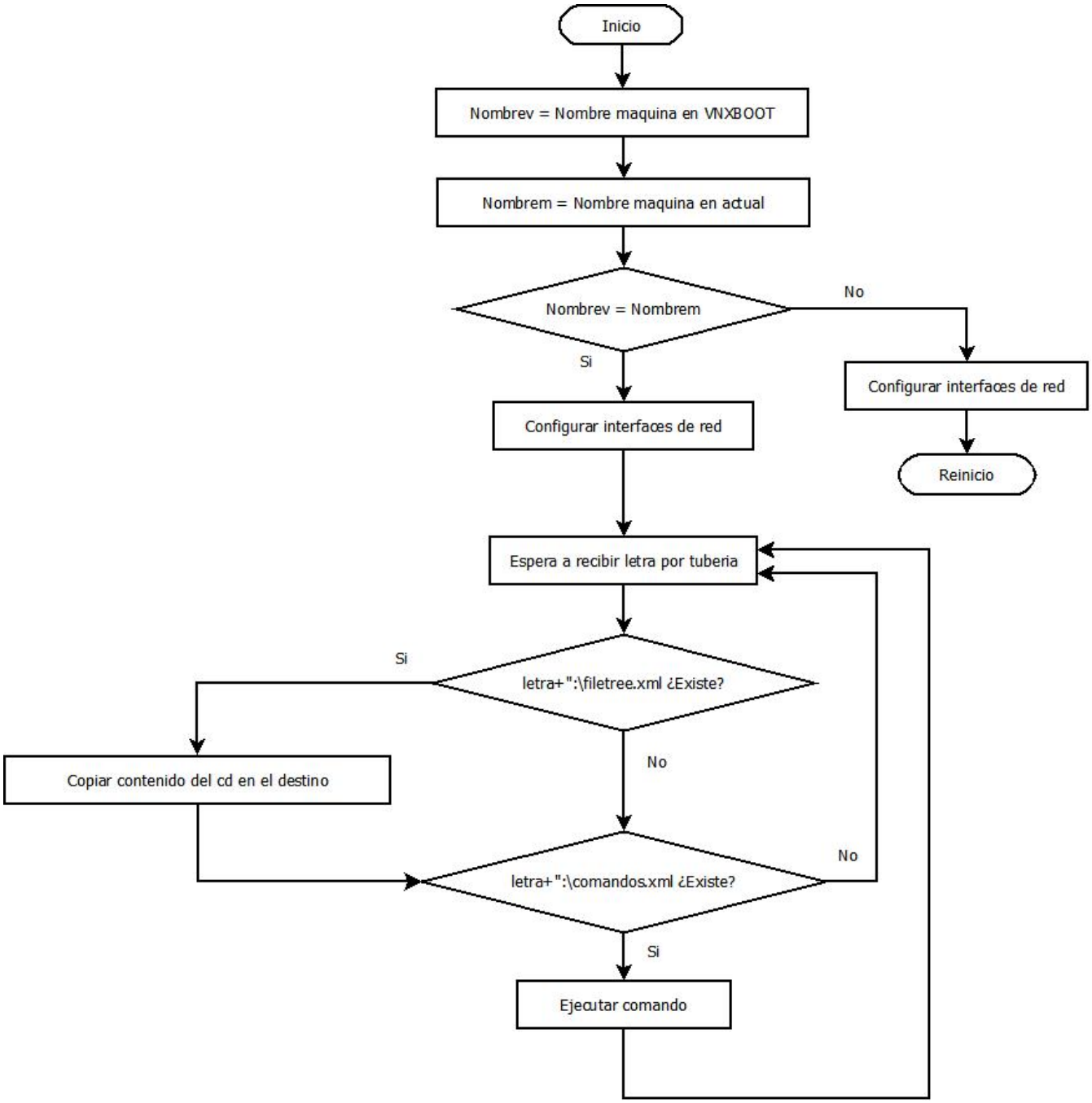


Fig. 15. Diagrama de flujo de la ejecución de VnxDaemon

3.2.3 VnxDaemon. Fases de la ejecución

Según la gráfica anterior, La ejecución del VnxDaemon se puede dividir más detalladamente en cuatro fases:

- Configuración del entorno.
- Configuración de la tubería.
- Espera de comandos.
- Ejecución de comandos.

Una vez que se ha configurado el sistema, lo cual implica las 2 primeras fases, VNXACE se queda esperando a la inserción de un cd con un formato correcto que contenga los ficheros necesarios para la ejecución de comandos o simplemente la copia de ficheros al sistema virtualizado.

En las siguientes secciones se detallará cada una de las cuatro fases en las que se puede dividir VnxDaemon.

3.2.3.1 Configuración del entorno.

En la primera fase de se encuentra la configuración del entorno del equipo virtualizado. En primer lugar debe leerse el fichero de configuración con formato XML que estará en el dispositivo lector de cd virtual. Este fichero se creó con anterioridad por VNX a través de las opciones que fueron configuradas en el fichero de especificaciones del entorno que se introduce como parámetro.

```
<?xml version="1.0" encoding="UTF-8"?>
<create_conf>
  <id>win7-Qck_U4</id>
  <vm name="win7">
    <filesystem
type="cow">/usr/share/vnx/filesystems/root_fs_win7</filesystem>
    <mem>1048576</mem>
    <kernel>default</kernel>
    <if id="1" net="Net0" mac=",02:fd:00:08:01:01" name="">
      <ipv4 mask="255.255.255.0">10.0.0.8</ipv4>
    </if>
    <o_flag></o_flag>
    <e_flag></e_flag>
    <Z_flag>1</Z_flag>
    <F_flag>0</F_flag>
    <notify_ctl>/tmp/vnx_notify.ctl.WIxCwT</notify_ctl>
  </vm>
</create_conf>
```

Listado 2. XML de definición de una máquina virtual para VNX

Una vez procesado el fichero de configuración, y cargado en memoria las opciones a configurar, se procede a cambiar el nombre de la máquina virtual. Una vez que se ha cambiado el nombre de la máquina virtual, el programa se encarga de reiniciar la máquina virtual. Con ello se consigue que los cambios queden implantados en la máquina virtual.

Una vez que se ha reiniciado se continua el proceso de configuración con los parámetros IP, Gateway y DNS según lo especificado en el XML de entrada. Una vez que se ha terminado de configurar la máquina virtual se procede a entrar en la siguiente fase, la de espera de comandos.

El código asociado para la configuración de los diferentes parámetros de la máquina virtual se ha incluido en el Apéndice C, Sección 10.3.2.

3.2.3.2 Configuración de la tubería.

En el correcto funcionamiento de todos los componentes del programa es necesario que haya comunicación entre los componentes de VnxClient y VnxDaemon. Para ello, en un primer lugar se hizo un estudio de las diferentes opciones que brindaba el entorno Windows en cuanto a comunicación entre procesos. Se eligió la comunicación a través de tuberías al no tener problemas de compatibilidad ni necesitar software, hardware o abrir un puerto en el sistema virtualizado con el que los posibles atacantes puedan intentar acceder.

Para poder obtener realimentación y seguimiento de eventos relacionados con VnxDaemon, se establecen dos tuberías, una en cada sentido. En una tubería por las que se enviaran la letra del cd virtual que se ha insertado en una tubería y en la otra de sentido opuesto los eventos producidos. A continuación se detallaran las funciones más importantes a tener en cuenta en el proceso de apertura de una tubería. Se ha de comentar que solo se muestra el proceso de la tubería que transporta la letra de la unidad al ser la tubería más crítica para VNXACE, siendo el proceso de apertura de la otra tubería análogo a esta.

Mediante el siguiente código se expone la apertura de la tubería en el lado de VnxDaemon con el nombre vnxletter. Este lado actúa de cliente siendo VnxClient el creador de esta tubería.

```
int receptionrx()
{
      .
      .
LPTSTR lpszPipename = TEXT("\\\\.\\pipe\\vnxletter");  // Nombre del
pipe
      .

            hPiperx = CreateFile(
                  lpszPipename,   // pipe name
                  GENERIC_READ |  // read and write access
                  GENERIC_WRITE,
                  0,              // no sharing
                  NULL,           // default security attributes
                  OPEN_EXISTING,  // opens existing pipe
                  0,              // default attributes
                  NULL);          // no template file

            // Break if the pipe handle is valid.

            if (hPiperx != INVALID_HANDLE_VALUE) {
                  dwMode = PIPE_READMODE_MESSAGE;
                  fSuccess = SetNamedPipeHandleState(
                        hPiperx,    // pipe handle
                        &dwMode,  // new pipe mode
                        NULL,     // don't set maximum bytes
                        NULL);    // don't set maximum time
                  if ( ! fSuccess)
                  {
                        return -1;
                  }
.
.
.
```

Listado 3. Apertura del PIPE en el lado de VnxDaemon

Como se puede observar en el código anterior, primero se ha de crear el descriptor de fichero de la tubería ya creada por VnxClient mediante el método de creación de fichero pasándole como parámetro el siguiente nombre de fichero: "`\\\\.\\pipe\\vnxletter`". Una vez obtenido el descriptor de fichero se han de establecer los permisos del pipe mediante la rutina *SetNamedPipeHandleState*.

Esta rutina tiene la siguiente descripción:

```
BOOL WINAPI SetNamedPipeHandleState(
  __in      HANDLE hNamedPipe,
  __in_opt  LPDWORD lpMode,
  __in_opt  LPDWORD lpMaxCollectionCount,
  __in_opt  LPDWORD lpCollectDataTimeout
);
```

Listado 4. Descripción de la rutina SetNamedPipeHandleState

Esta rutina admite 4 parámetros diferentes que se explicaran a continuación:

- *hNamedPipe*: Descriptor de fichero que se ha creado con la función *CreateFile*.
- *lpMode*: Modo de apertura del Pipe. En este caso es el siguiente *PIPE_READMODE_MESSAGE* ya que se intercambian mensajes y no bytes.
- *lpMaxCollectionCount*: Número máximo de bytes que se han de recolectar antes de enviarlo al servidor. Este parámetro es necesario que sea *NULL* en caso que sea el servidor como es este caso.
- *lpCollectDataTimeout*: Tiempo máximo en milisegundos antes de enviar los datos por la red. Este parámetro es necesario que sea *NULL* en caso que sea el servidor como en este caso.

Una vez que se ha realizado la configuración del Pipe, se queda a la espera de recibir datos por esta tubería para ser procesada.

3.2.3.3 Espera de comandos

En esta fase, el programa VnxDaemon se queda esperando a la recepción desde la tubería de la letra de una unidad que se haya insertado. Una vez que VnxClient ha enviado la letra, VnxDaemon lo recoge y la procesa entrando en la siguiente fase.

Utilizando el descriptor de fichero que está relacionado con la tubería, obtenido de la fase anterior, se procede a la lectura de los datos que hay en el interior mediante la rutina *ReadFile*. Una vez leídos los datos de la tubería sin ningún error, se procede a llamar a la función de *insercioncd* pasándole como parámetro los datos obtenidos. El siguiente código es un extracto de lo comentado en el que solo se ha reflejado la función más relevante.

```
while (1)
{
do
    {
// Read from the pipe.
string letra = "D";
DWORD cbToRead = (lstrlen((LPCWSTR)StringToWString(&letra).c_str())+1...
fSuccess = ReadFile(
hPiperx,    // pipe handle
chBuf,    // buffer to receive reply
     BUFSIZE*sizeof(TCHAR),  // size of buffer
     (LPDWORD) &cbToRead,  // number of bytes read
     //(LPDWORD) 8,
     NULL);    // not overlapped

    if ( ! fSuccess && GetLastError() != ERROR_MORE_DATA )
     break;
                                        .
```

```
                                        .
                                        .
switch(dwWaitResult){
case WAIT_OBJECT_0:
 // Mandamos la letra al metodo insercioncd
 // que se encargará de procesarla.
 insercioncd(s);
 ReleaseMutex(hMutex);
default:
 ;
}
Sleep(2000);
```

Listado 5. Lectura del pipe.

Cabe destacar que en la realización de todas las partes se han utilizado *Mutex* para evitar que varios hilos lanzados de forma consecutiva por el sistema operativo puedan producir un mal funcionamiento del programa en cuestión.

El código asociado a estas fases se ha incluido en el apartado C, Sección 7.3.2.

3.2.3.4 Ejecución de comandos

En la fase de ejecución de comandos se puede dividir en dos sub-fases claramente diferenciadas pero interrelacionadas.

- Copia de ficheros de la maquina host a la máquina virtual
- Ejecución de comandos.

Es recomendable resaltar que en el XML que se comparte de VNX pueden estar definidas las dos acciones o una de las dos ya que es independiente su ejecución una de otra.

Si se quiere que se ejecute el *filetree* y los comandos en un mismo paso se deben introducir el mismo identificador en ambos, sin embargo, es importante recalcar que siempre se realizará la copia antes de la ejecución de comandos.

Las definiciones de los XMLs y la sintaxis de ejecución serán detalladas en la sección dedicada a la función de Ejecución de comandos de VNX, siendo esta la sección numerada con 3.3.1.2.

3.2.3.4.1 Copia de ficheros

En la fase de copia de ficheros, la maquina host deberá incluir los ficheros a copiar en la imagen del cd que se le inserta a la máquina virtual. Adicionalmente, en el fichero XML habrá sentencias que determinen el destino del directorio a copiar en la máquina virtual.

Una vez que se monte la unidad de cds virtual en la máquina virtual, el programa procesará el fichero XML y copiara los ficheros a la ruta destino. Cuando esto se haya realizado, VNXACE envía el carácter "1" a VNX para indicarle de su finalización.

3.2.3.4.2 Ejecución de comandos

Para poder ejecutar comandos en las máquinas virtuales, primero se han de tener definidas etiquetas *exec* en el xml del escenario. Una vez que se han definido los comandos que se pueden ejecutar en la máquina virtual, se procederá a invocar el comando de ejecución de comandos en la máquina virtual. Este procedimiento se hará con la siguiente instrucción que se ha de lanzar desde la maquina host.

```
vnx –f fichero del escenario.xml -x nombrecomando
```

Listado 6. Comando a ejecutar para la ejecución de un determinado comando.

Este comando desembocará en la creación del fichero comandos.xml, la creación de una imagen de CD y su inserción en la máquina virtual correspondiente. Esta fase se detallará más adelante en la sección de integración.

Una vez que se ha insertado el CD, la aplicación VnxClient capturará el evento de inserción del CD y enviará a VnxDaemon, mediante la tubería establecida en el momento de inicio, la letra correspondiente a la unidad con la imagen nueva.

VnxDaemon comprobará la existencia del fichero comandos.xml y lo procesará ejecutando los comandos en orden de inserción en el xml del escenario.

Cuando se termina la ejecución de todos los comandos en esa máquina virtual, el VnxDaemon manda un "1" por el puerto serie avisando así al VNX que ha terminado de ejecutar todos los comandos y que puede pasar o bien a la definición de un nuevo comandos.xml para otra máquina virtual o si no hay más, finalizar su ejecución.

Para simplificar en tareas repetitivas en las que se debe realizar lo mismo en todas las máquinas virtuales como puede ser, el mantenimiento de estas, el mismo nombre de comando puede ser definido a una o varias máquinas virtuales, dando por resultado que

se hará un ciclo con su creación de comandos.xml para cada máquina sin tener que ejecutar otra vez la línea para cada máquina.

El extracto de código referente a la ejecución de comandos se puede consultar en el Apéndice C. Sección 10.3.5.

3.2.4 VnxClient. Fases de la ejecución

Como se ha comentado en apartados anteriores, la parte de VnxClient que es ejecutada con los permisos del usuario que se ha autenticado en el sistema, tiene la función de registrase en el manejador de eventos de Windows para recibir los eventos de inserción de CDs. Además, contiene funcionalidades añadidas de contener un registro de eventos producidos, ya sea por el manejador de eventos como por VnxDaemon.

Las fases de ejecución de VnxClient son las siguientes:

- Registro del manejador de eventos
- Creación de la tubería
- Espera de eventos
- Envío de los datos a VnxDaemon

3.2.4.1 Registro del manejador de eventos

Para que el VNXACE funcione correctamente, es necesario que la parte cliente se registre en el entorno de Windows para ser capaz de recibir los eventos de inserción de CD.

Para registrar un cliente para recibir los eventos se debe usar el método *RegisterDeviceNotificationA* proporcionado por la API de Windows (11). La descripción del método es la siguiente.

```
HDEVNOTIFY WINAPI RegisterDeviceNotification(
  __in  HANDLE hRecipient,
  __in  LPVOID NotificationFilter,
  __in  DWORD Flags
);
```

Listado 7. Descripción del método de registro de eventos.

Esta función recibe 3 parámetros:

- *hRecipient*: Un manejador de ventanas donde pueda recibir los eventos de los dispositivos indicados. En nuestro caso, creamos una ventana nueva para recibir los eventos.
- *NotificationFilter*: Puntero hacia una estructura de datos de tipo *DEV_BROADCAST_DEVICEINTERFACE_A* donde se indica que tipo de dispositivos se desea escuchar los eventos.
- *Flags*: Este parámetro es binario por lo que se puede insertar varios *Flags* al mismo tiempo. En este caso se ha definido que es *DEVICE_NOTIFY_WINDOW_HANDLE* al ser una ventana creada anteriormente donde se recibirá los eventos y *DEVICE_NOTIFY_ALL_INTERFACE_CLASSES*, obteniéndose así todos los eventos producidos por cualquier dispositivo. La razón por la cual se escucha todos los dispositivos y no se filtra solo los CDs mediante el parámetro *dbcc_classguid* de la estructura *DEV_BROADCAST_DEVICEINTERFACE_A* es por evitar errores próximos si en las próximas versiones de *Libvirt* modificaban el driver que controla el *CDROM* y no esté contemplado VnxDaemon.

A continuación se expondrá el código fuente de la función que se encarga del registro del cliente en el manejador de eventos de Windows, resaltando las llamadas importantes.

Es importante recalcar que para que el servicio funcione, debe estar asociado a una pantalla que tenga permisos de administrador. En el caso de Windows XP, esta pantalla es el propio escritorio, sin embargo, por las restricciones que impone Windows 7, esta pantalla es ejecutada en segundo plano siendo mostrada solo en caso que la aplicación ejecutada necesite entorno gráfico.

```
int registro()
{
	.
	.
	.
	const char *className = "VnxClientGui";
	WNDCLASSA wincl = {0};
	wincl.hInstance = GetModuleHandle(0);
	wincl.lpszClassName = className;
	// La funcion donde vamos a tratar el evento
	wincl.lpfnWndProc = WinProc;
```

```
if (!RegisterClassA(&wincl))
{
      DWORD le = GetLastError();
      return 1;
}//if
.
.
.
// Creamos un entorno donde poder registrar el evento
HWND hwnd = CreateWindowExA(WS_EX_TOPMOST, className, className,
      0, 0, 0, 0, 0, parent, 0, 0, 0);
if (!hwnd)
{
      DWORD le = GetLastError();
      return 1;
}//if
// Definimos que tipo de eventos y de donde su procedencia
// en este caso el origen es la unidad de CDs


DEV_BROADCAST_DEVICEINTERFACE_A notifyFilter = {0};
notifyFilter.dbcc_size = sizeof(notifyFilter);
notifyFilter.dbcc_devicetype = DBT_DEVTYP_DEVICEINTERFACE;
HDEVNOTIFY   hDevNotify   =     RegisterDeviceNotificationA(hwnd,
&notifyFilter,       DEVICE_NOTIFY_ALL_INTERFACE_CLASSES       |
DEVICE_NOTIFY_WINDOW_HANDLE);
//    DEVICE_NOTIFY_SERVICE_HANDLE);
if (!hDevNotify)
{
      DWORD le = GetLastError();
 .
 .
 .
```

Listado 8. Registro del cliente en el manejador de eventos

3.2.4.2 Creación de la tubería

Para poder enviar los eventos hacia VnxDaemon se necesita crear la tubería, como en este caso VnxClient es el emisor de eventos, este actuará como servidor.

Para crear un Pipe en servidor es necesario llamar a la función *CreateNamedPipe* (12). Esta función tiene la siguiente descripción:

```
HANDLE WINAPI CreateNamedPipe(
  __in      LPCTSTR lpName,
  __in      DWORD dwOpenMode,
  __in      DWORD dwPipeMode,
  __in      DWORD nMaxInstances,
  __in      DWORD nOutBufferSize,
  __in      DWORD nInBufferSize,
  __in      DWORD nDefaultTimeOut,
  __in_opt  LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

Listado 9. Descripción de CreateNamedPipe

Esta función recibe 8 parámetros siguientes:

- *lpName*: Nombre que se va a dar al pipe.
- *dwOpenMode*: Modo de apertura del Pipe.
- *dwPipeMode*: Modo de la tubería.
- *nMaxInstances*: Número máximo de instancias que pueden crearse de la tubería.
- *nOutBufferSize*: Numero de bytes reservado para el buffer de salida.
- *nInBufferSize*: Numero de bytes reservado para el buffer de entrada.
- *nDefaultTimeOut*: Valor del *time-out* por defecto. En caso de que sea 0, el valor por defecto es 50 milisegundos.
- *lpSecurityAttributes*: Puntero a una estructura tipo SECURITY_ATTRIBUTES en el que se especifica las opciones de seguridad del Pipe.

A continuación se incluye el extracto de código donde se procede a la apertura del pipe en el que se incluyen los parámetros definidos para su utilización por VnxClient y VnxDaemon.

```
BOOL   fConnected = FALSE;
      DWORD  dwThreadId = 0;
      LPTSTR lpszPipename = TEXT("\\\\.\\pipe\\vnxletter");

      // Creamos el Pipe
      hPipewrite = CreateNamedPipe(
            lpszPipename,             // pipe name
            PIPE_ACCESS_DUPLEX,       // read/write access
            PIPE_TYPE_MESSAGE |       // message type pipe
            PIPE_READMODE_MESSAGE |   // message-read mode
            PIPE_WAIT,                // blocking mode
            PIPE_UNLIMITED_INSTANCES, // max. instances
            BUFSIZE,                  // output buffer size
            BUFSIZE,                  // input buffer size
            0,                        // client time-out
            NULL);                    // default security attribute

      if (hPipewrite == INVALID_HANDLE_VALUE)
      {
            .
            .
            .
      }
      return 0;
```

Listado 10. Creación de la tubería

3.2.4.3 Espera de eventos

Una vez que se ha registrado VnxClient en el manejador de eventos para recibir los eventos producidos por la inserción de *CDs* y que también se ha procedido a la creación de la tubería para la conexión con VnxDaemon, VnxClient se queda a la espera de algún evento para poder procesarlo y enviarlo a VnxDaemon.

Cuando algún evento se produce, el manejador de Windows llama automáticamente al método que se le haya configurado para que responda de forma automática, en este caso, se registró el método *WinProc*, por lo que cuando se produce un evento, Windows llama automáticamente a *WinProc* pasándole más información sobre el evento a través del parámetro *wParam*.

Este método analiza la naturaleza del evento a través del parámetro *wParam*, solo actuando en el caso que el evento sea de tipo *DBT_DEVICEARRIVAL*. Para obtener la letra de la unidad responsable del evento, se ayuda de la función *FirstDriveFromMask*, la cual se puede estudiar su código en el Apéndice C sección 7.3.6 de esta memoria.

```
LRESULT CALLBACK WinProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
	.
	.
	.
	PDEV_BROADCAST_VOLUME lpdbv;
	PDEV_BROADCAST_HDR lpdb = (PDEV_BROADCAST_HDR)lParam;
	.
	.
	.
		switch(wParam)
		{
		// Caso de llegada de dispositivo nuevo (HD, CD, DVD,..)
		case DBT_DEVICEARRIVAL:
			lpdbv = (PDEV_BROADCAST_VOLUME)lpdb;
			// Vemos de que letra se trata el nuevo dispositivo
			cd = FirstDriveFromMask(lpdbv ->dbcv_unitmask);
			temp << cd;
			temp >> comando;
			// Convertimos todo a CString
			commandc.SetString((StringToWString(comando.c_str())
			).c_str());
			sendletter(commandc);
			break;
	.
	.
	.
	return 1;
}
```

Listado 11. Procedimiento de envío de los datos a VnxDaemon.

3.2.4.4 Envío de los datos a VnxDaemon

Una vez que se ha recibido y procesado el evento, se procede a enviarlo a través de la tubería que ha abierto VnxDaemon.

Para ello se realiza en primer lugar la conexión con el Pipe a través de la función *ConnectNamedPipe*, el cual se le pasa por parámetros el descriptor de fichero de la tubería que se ha obtenido de la llamada a la función de *CreateNamedPipe*.

Una vez conectada la tubería, se procede a escribir dentro de ella tratándola como si fuera un fichero a través de la función *WriteFile*.

En el extracto del código que se incluye a continuación se muestra el procedimiento de escritura, resaltando las funciones más importantes.

```
fConnected = ConnectNamedPipe(hPipewrite, NULL) ?
TRUE : (GetLastError() == ERROR_PIPE_CONNECTED);

      if (fConnected)
      {
            DWORD cbToWrite =
(lstrlen((LPCWSTR)(lettercs.GetString()))+1)*sizeof(TCHAR);
            // Escribimos la letra por el pipe
            fSuccess = WriteFile(
                  hPipewrite,
                  lettercs.GetString(),
                  cbToWrite,
                  (LPDWORD) &cbWritten,
                  NULL);
```

Listado 12. Escritura en el Pipe

Una vez que se ha realizado el procedimiento de envío de la letra a VnxDaemon, el cliente vuelve al estado de espera preparado para el próximo evento que le pueda llegar a través del manejador de Windows.

Adicionalmente a este cliente se le ha incluido la funcionalidad de registro de todos los eventos que se producen, ya sea de origen VnxClient o VnxDaemon. Para poder registrar los eventos producidos por VnxDaemon, se incluyó una segunda tubería totalmente independiente a la de envío de la letra para no influir en el funcionamiento principal del sistema. Esta tubería por donde VnxClient recibe los eventos de VnxDaemon está registrada al contrario ya que es VnxDaemon quien crea esta tubería y VnxClient actúa de cliente.

3.3 Integración

Como se ha comentado en el capítulo del estado del arte correspondiente a VNX. Este está separado en dos partes bien diferenciadas entre las que se comunican a través de una API común.

Para que se pudiera desarrollar de forma correcta, se necesitó implementar las funciones que estaban marcadas en la API y alineándose con el desarrollo necesario para que también pudiera comunicarse con VNXACE para el correcto funcionamiento.

A continuación se proceden a listar las funciones de la API así como su descripción y sus acciones en el caso que se virtualice Windows. Adicionalmente se explicará detalle aquellas funciones cuya implementación necesite de pasos adicionales en el caso que el sistema virtualizado sea Windows.

Función	Descripción
defineVM	Se explicará de forma detallada en el apartado 3.3.1
undefineVM	Mediante esta llamada se elimina la máquina virtual del hipervisor, esta función no necesita casos especiales en el caso que la máquina virtual fuera un Windows ya que no requiere interacción con ella.
destroyVM	Mediante esta llamada se elimina la máquina virtual del hipervisor, esta función no necesita casos especiales en el caso que la máquina virtual fuera un Windows ya que no requiere interacción con ella.
startVM	Mediante la llamada a esta función, VNX solicita a libvirt que arranque la máquina virtual definida mediante el comando defineVM. Esta interacción no requiere de pasos especiales en caso que el sistema operativo sea Windows ya que no requiere de la interacción con este.
shutdownVM	Mediante la llamada a shutdownVM, VNX solicita a libvirt que apague de forma ordenada la máquina virtual mediante el envío de las correspondientes señales ACPI. Esta función no necesita pasos especiales en caso que la máquina virtual ejecute un sistema operativo Windows.

Función	Descripción
saveVM	Mediante esta función, VNX solicita a libvirt los datos de configuración de la máquina virtual para que sean guardados. Una vez realizado, se solicita a libvirt el pausado de la máquina virtual. Esta función no necesita pasos especiales en caso que la máquina virtual ejecute un sistema operativo Windows.
restoreVM	Mediante esta función, VNX solicita a libvirt que restaure una máquina virtual anteriormente pausada. Este procedimiento no hace perder los cambios producidos una vez que se haya iniciado la máquina. Esta función no necesita pasos especiales en caso que el sistema operativo de la máquina virtual sea Windows.
SuspendVM	Mediante esta función, VNX solicita a libvirt la suspensión temporal de una máquina virtual. Esta función no necesita pasos especiales en caso que el sistema operativo de la máquina virtual sea Windows.
resumeVM	Mediante esta función, VNX solicita a libvirt la reanudación de la máquina virtual anteriormente suspendida. Esta función no necesita pasos especiales en caso que el sistema operativo de la máquina virtual sea Windows.
rebootVM	Mediante esta función, VNX solicita a libvirt el reinicio ordenado de la máquina virtual mediante el envío de la correspondiente señal ACPI. Dicha función no necesita pasos especiales en caso que sea Windows el sistema operativo virtualizado.
resetVM	Mediante esta función, VNX solicita a libvirt el reinicio inmediato de la máquina virtual. Dicha función no necesita pasos especiales en caso que sea Windows el sistema operativo virtualizado.
executeCMD	Se explicará de forma detallada en el apartado 3.3.2

3.3.1 Funciones especiales

3.3.1.1 defineVM

Dicha función realiza los ficheros asociados a la máquina virtual, así como el fichero QCOW2 donde se guardaran las diferencias entre la imagen del sistema operativo madre y las diferencias en esta por la ejecución del sistema operativo.

En este paso, además de crearse los ficheros para permitir la ejecución del sistema operativo mediante libvirt, se crea el fichero de inicialización para VNXACE que es el utilizado para auto configurar la máquina Windows.

Un ejemplo del fichero de autoconfiguración es el siguiente.

```
<?xml version="1.0" encoding="UTF-8"?>
<create_conf>
  <id>win7-Qck_U4</id>
  <vm name="win7">
    <filesystem
type="cow">/usr/share/vnx/filesystems/root_fs_win7</filesystem>
    <mem>1048576</mem>
    <kernel>default</kernel>
    <if id="1" net="Net0" mac=",02:fd:00:08:01:01" name="">
      <ipv4 mask="255.255.255.0">10.0.0.8</ipv4>
    </if>
    <o_flag></o_flag>
    <e_flag></e_flag>
    <Z_flag>1</Z_flag>
    <F_flag>0</F_flag>
    <notify_ctl>/tmp/vnx_notify.ctl.WIxCwT</notify_ctl>
  </vm>
</create_conf>
```

Listado 13. XML de definición de una máquina virtual para VNX

Una vez que se ha creado el fichero xml anterior, este se introduce en un fichero imagen de un CD (.iso) para su posterior vinculación con la máquina virtual.

Adicionalmente se crea el fichero de configuración para que cuando se ejecute el comando startVM sea incluido dentro este.

Un ejemplo del fichero es el siguiente.

```
<?xml version="1.0" encoding="UTF-8"?>
<domain type="kvm">
  <name>win7</name>
  <memory>1048576</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch="i686">hvm</type>
    <boot dev="hd"/>
    <boot dev="cdrom"/>
```

```
  </os>
  <features>
    <pae/>
    <acpi/>
    <apic/>
  </features>
  <clock sync="localtime"/>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type="file" device="disk">
      <source
file="/root/.vnx/scenarios/simple_w7/vms/win7/fs/root_cow_fs"/>
      <target dev="hda"/>
      <driver name="qemu" type="qcow2"/>
    </disk>
    <disk type="file" device="cdrom">
      <source
file="/root/.vnx/scenarios/simple_w7/vms/win7/fs/opt_fs.iso"/>
      <target dev="hdb"/>
    </disk>
    <interface type="bridge" name="eth1" onboot="yes">
      <source bridge="Net0"/>
      <mac address="02:fd:00:08:01:01"/>
    </interface>
    <graphics type="vnc" listen=""/>
    <serial type="unix">
      <source mode="bind"
path="/root/.vnx/scenarios/simple_w7/vms/win7/win7_socket"/>
      <target port="1"/>
    </serial>
  </devices>
</domain>
```

Listado 14. XML de definición de una máquina virtual en libvirt

Adicionalmente en todas las máquinas virtuales Windows, se crea un socket sobre el puerto serie por el cual se enviarán información relativa a la ejecución de comandos.

En dicho socket, cuando una operación de ejecución de comandos o de copia de ficheros se finalice, se recibirá un "1" el cual es enviado por VNXACE para informar que ha finalizado correctamente. Para posteriores versiones está programado añadir más funcionalidad a este socket.

3.3.1.2 executeCMD

Mediante esta función, VNX solicita a libvirt la ejecución de determinados comandos dentro de la máquina virtual, adicionalmente mediante dicha función se realiza el copiado de un sistema de ficheros de ordenado anfitrión al virtualizado. Esta función depende del sistema operativo que este virtualizado, por lo que a continuación se exponen los pasos en el caso concreto del sistema operativo Windows.

En primer lugar se ha de preparar el XML con la definición del comando que se quiera ejecutar en la máquina virtual y situarlo en una carpeta temporal.

Este XML puede contener entradas para la copia de ficheros delimitadas con las etiquetas <filetree> y para la ejecución de comandos con la etiqueta <exec>.

Un ejemplo del XML definiendo la inclusión de ficheros en la máquina virtual es el siguiente:

```
<command>
<id>win7-qtpyl5</id>
<filetree seq="vnxtxt" root="c:\temp">conf/txtfile</filetree>
<exec seq="vnxtxt" type="verbatim" mode="system">start /max notepad
c:\temp\vnx.txt</exec>
</command>
```

Listado 15. XML de definición de un comando.

En dicho XML se le informa que cuando se ejecute los comandos con el identificador *vnxtxt*, copie todos los ficheros en la ruta c:\temp. Los parámetros admitidos en el *filetree* se detallan en la siguiente lista.

Parámetro	Descripción
Seq	Identificador de cada acción. Utilizado para llamar a la acción desde VNX
root	Destino de los ficheros en la máquina virtual
Contenido	El origen del fichero. En esta versión no es utilizado permitiendo su uso para desarrollos futuros.

Para ejemplificar la ejecución de comandos se hará con otro ejemplo de un escenario con etiquetas *exec*:

```
<vm name="uml2" type="libvirt-kvm-windows">
<filesystem type="cow">/usr/share/vnuml/filesystems/root_fs_winxp</filesystem>
   <mem>256M</mem>
   <if id="1" net="Net0">
     <ipv4>10.0.0.2</ipv4>
   </if>
   <route type="ipv4" gw="10.0.0.3">default</route>
   <filetree seq="vlc" root="c:\temp1">/home/<usuario>/entregable</filetree>
   <exec seq="vlc" type="file">/home/<usuario>/lista.xml</exec>
   <exec seq="dir" type="verbatim" mode="system">dir </exec>
 </vm>
```

Listado 16. XML de configuración interna

En este ejemplo se puede ver dos líneas *exec*, una de tipo *file* y otra de tipo *verbatim.*

La instrucción de tipo *file* hará que se integre el contenido de /home/<usuario>/lista.xml dentro del fichero comandos.xml. El fichero *Lista.xml* ha de ser el mismo formato que comandos.xml ya que se va a añadir el contenido a comandos.xml. Un ejemplo del fichero lista.xml puede ser el siguiente.

```
<exec seq="vlc" type="verbatim" mode="processn">
 c:\temp1\PresentationCD\PPTVIEW.EXE /L "c:\temp1\PresentationCD\playlist.txt"
</exec>
<exec seq="vlc" type="verbatim" mode="processy">
 c:\temp1\sleep.exe 5
</exec>
<exec seq="vlc" type="verbatim" mode="processy">
 taskkill /IM PPTVIEW.exe
</exec>
<exec seq="vlc" type="verbatim" mode="processy">
 c:\temp1\smplayer_portable_0.6.9\smplayer.exe -close-at-end -fullscreen
 c:\temp1\video.mp4
</exec>
```

Listado 17. XML de ejecución y copia de comandos

Respecto a la instrucción de tipo *verbatim,* que se incluirá sin modificación alguna en el fichero comandos.xml, tiene varios modificadores configurando así el modo de ejecución deseado: *mode* o *gui*.

- El modificador *mode* admite las siguientes opciones.

Opciones	Descripción
system	Se ejecuta como si se hubiera tecleado la instrucción final en una consola de comandos.
processy	Te crea un proceso hijo y se bloquea no pudiendo seguir con el siguiente comando hasta que el proceso no haya terminado.
processn	Te crea un proceso hijo y no se bloquea pudiendo seguir con la siguiente instrucción en ejecución de comandos.

- El modificador *gui* es un modificador opcional y pueden tener los valores *yes* y *no.*

Esto permite que si se ejecuta alguna aplicación gráfica se muestre en el escritorio de la máquina virtual o no. Por defecto si no se pone la opción esta se mostraría. En Windows 7 para evitar bloqueos, se recomienda que se defina la opción a no.

Adicionalmente al fichero XML, se incluye en esta carpeta temporal los directorios que se quieran incluir en la máquina virtual definiendo en el XML la ruta a donde se quieren copiar.
Dicho directorio temporal con el XML y los datos que se desean copia, se introduce en una imagen .ISO para que posteriormente pueda ser incluido en la máquina virtual mediante libvirt.
Este proceso se realiza creando otro XML en el que se define diversos parámetros necesarios para que libvirt pueda adjuntar el ISO a la máquina virtual. Un ejemplo de dicho XML puede ser el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<disk type="file" device="cdrom">
  <driver name="qemu" cache="default"/>
  <source file="/tmp/disk.TL_4JI.iso"/>
  <target dev="hdb"/>
  <readonly/>
</disk>
```

Listado 18. XML de definición del ISO necesario para la ejecución de comandos

Una vez que se ha incluido el fichero en la máquina virtual, VNX se queda esperando a la terminación de la ejecución por parte de VNXACE si así ha sido definido en el XML de definición del entorno mediante la opción *mode* que se explicó en apartados anteriores.

Esta espera finaliza en el momento el que VNXACE envía mediante su puerto serie el carácter “1”. En ese momento, se procede a eliminar todos los ficheros temporales que se hayan creado para dicho procedimiento.

Virtualización de routers Cisco mediante Dynamips

That's all it is: information. Even a simulated experience or a dream; simultaneous reality and fantasy. Any way you look at it, all the information that a person accumulates in a lifetime is just a drop in the bucket.

Ghost in the Shell (1995)

4 Virtualización de routers Cisco mediante Dynamips.

4.1 Introducción

Uno de los objetivos de VNX es poder utilizarse para la creación de *honeynets* de forma dinámica, construyendo así una red heterogénea lo más fiable a una red de producción. En esta red debería de estar compuesta por estaciones terminales y routers de diferentes fabricantes entre otros dispositivos.

Sin embargo, hasta este momento, VNX solo permite la virtualización de Linux y Windows mediante la librería de virtualización libvirt. Esta librería, al virtualizar hardware de un ordenador de escritorio, no permite la emulación de routers ni de otros dispositivos que dispongan de hardware más especializado.

Para solucionar este inconveniente, se ha utilizado el software de virtualización *Dynamips* que emula la arquitectura hardware de diversos modelos de routers Cisco. Mediante este software de virtualización es posible crear routers Cisco virtualizando su sistema operativo (iOS) y con ello poder integrarlos dentro de las *honeynets*.

Sin embargo, este sistema operativo es propietario de Cisco y solo puede ser utilizado en el caso que se posea su licencia.

En este capítulo se procederá a describir el proceso de integración que se ha realizado de la herramienta de *Dynamips* a VNX para habilitar la posibilidad de virtualización de routers de la marca Cisco.

4.2 Instalación

Existen versiones de *Dynamips* para Linux, Windows y Mac pero como se va a usar conjuntamente con VNX, se utilizará la versión disponible para la plataforma Linux.

La instalación se hace a través de cualquier gestor de paquetes. En el caso de Ubuntu se instalaría a través de *aptitude*, *synaptic o apt-get* al estar presente en los repositorios *Debian*:

```
$ sudo apt-get install dynamips
```

Listado 19. Comando de instalación de dynamips

Para otras versiones, se debe dirigir a la web del desarrollador: http://sourceforge.net/projects/dyna-gen/files/

4.3 Configuración

La configuración de *Dynamips* se hace mediante ficheros de texto debidamente formateados para que puedan ser procesados correctamente por el programa.

Un ejemplo de un fichero de configuración puede ser el siguiente:

```
[localhost]
 [[7200]]
  image = \Program Files\Dynamips\images\c7200-jk9o3s-mz.124-7a.image
  #On Linux / Unix use forward slashes:
  #image = /opt/7200-images/c7200-jk9o3s-mz.124-7a.image
  npe = npe-400
  ram = 160
 [[ROUTER R1]]
  s1/0 = R2 s1/0
 [[ROUTER R2]]
  # No need to specify an adapter here, it is taken care of
  # by the interface specification under Router R1
```

Listado 20. Ejemplo de configuración de Dynamips

En la primera línea se define en que máquina es donde se va a lanzar la emulación, y por tanto, donde está instalado *Dynamips*, en este caso en *localhost*.

En segundo lugar se define qué modelo de hardware Cisco se quiere emular, en este caso un 7200. Dentro de esta sección, se configura la ruta donde se puede encontrar el firmware necesario para la ejecución del emulador, adicionalmente se indica la RAM que va a disponer el hardware emulado. Después se procede a definir los routers que estarán emulados, en el caso del ejemplo, serán dos routers emulados conectados a través de una línea serie.

4.4 Ejecución

Esta herramienta puede ser ejecutada de dos formas diferentes.

- Mediante línea de comandos.

Este modo permite crear un *router* simplemente llamando al software *dynamips* con los modificadores adecuados.

- Modo hipervisor (con el modificador -H).

Con este modo te crea un demonio o hipervisor que corre en el puerto especificado en la línea de comandos. Ejemplo de la ejecución de un hipervisor en el puerto 7300:

```
dynamips -H 7300
```

Listado 21. Comando de ejecución de dynamips en modo demonio

Este modo es el que más potencia tiene, ya que se permite parar, encender y configurar varios *routers* en un mismo escenario. Para ello se utilizan sentencias que se envían a través del puerto especificado. Como ejemplo, se puede conectar a través de telnet al puerto 7300 y escribir la sentencia `hypervisor cmd_list vm`, que muestra los comandos que pueden ser introducidos desde el módulo vm.

4.5 Definición del escenario con routers Cisco.

Para realizar un escenario con *routers* cisco, lo primero que se tiene que hacer es tener una imagen del sistema operativo de Cisco de alguno de los *routers* que *dynamips* soporta y copiarlo en la ruta por defecto donde se tengan todas las imágenes de sistemas operativos virtuales. En los siguientes ejemplos se ha realizado con el modelo 3640.

Para explicar cómo se debe de definir un *router* cisco dentro de VNX.

```
<?xml version="1.0" encoding="UTF-8"?>
<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-1.93.xsd">
  <global>
    <version>1.92</version>
    <scenario_name>simple_dynamips</scenario_name>
    <automac offset="3"/>
    <vm_mgmt type="none" />
    <vm_defaults exec_mode="mconsole">
    </vm_defaults>
    <dynamips_ext>simple_dynamips-dn.xml</dynamips_ext>
  </global>
  <net name="Net0" mode="virtual_bridge" />
  <!-- NODES -->
  <vm name="R1" type="dynamips" subtype="3600" os="">
       <filesystem type="cow">/usr/share/vnx/filesystems/c3640</filesystem>
       <mem>256M</mem>
       <if id="1" net="Net0" name="e0/0">
              <ipv4>10.1.1.4/24</ipv4>
              <ipv6>2001:db8::1/64</ipv6>
    </if>
    <exec seq="brief" type="verbatim">show ip interface brief</exec>
    <exec seq="reload" type="verbatim">reload /home/<user>/R1.txt</exec>
  </vm>
  <vm name="R2" type="dynamips" subtype="3600" os="">
       <filesystem type="cow">/usr/share/vnx/filesystems/c3640</filesystem>
       <mem>256M</mem>
       <if id="1" net="Net0" name="e0/0">
              <ipv4>10.1.1.3/24</ipv4>
    </if>
    <exec seq="brief" type="verbatim">show ip interface brief</exec>
    <exec seq="file" type="file">/home/<user>/fichero.cmd</exec>
  </vm>
  <host>
    <hostif net="Net0">
       <ipv4>10.1.1.1/24</ipv4>
    </hostif>
  </host>

</vnx>
```

Listado 22. XML de definición de escenario.

Como se puede ver en el ejemplo, para especificar que una máquina virtual es un *router* se debe de poner en el parámetro *type* la etiqueta "*dynamips*" y en *subtype*, la familia del *router* cisco que queremos simular, en nuestro caso es de una familia 3600 dejando la etiqueta os en blanco pudiéndose utilizar en futuras actualizaciones.

En *filesystem* se ha de poner la imagen del sistema operativo Cisco IOS del modelo en concreto que se quiere simular.

Como el XML del VNX solo tiene como misión dar información del escenario, se ha de crear un XML aparte que tenga la definición característica de cada uno de los *routers*. Este XML se define en el VNX mediante esta línea: `<dynamips_ext>simple_dynamips-dn.xml</dynamips_ext>`

Se puede poner como ruta absoluta empezando por /, o como ruta relativa desde la carpeta de la definición del escenario.

```
<?xml version="1.0" encoding="UTF-8"?>

<vnx_dynamips xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/dynamips.xsd">

  <global>
    <sparsemem>true</sparsemem>
    <ghostios>false</ghostios>
    <hw>
       <slot id="0">NM-4E</slot>
       <slot id="1">NM-4T</slot>
       <console_base port="1000"/>
     </hw>
  </global>

  <vm name="R1">
     <hw>
       <chassis>c3640</chassis>
       <slot id="0">NM-4E</slot>
       <slot id="1">NM-4T</slot>
       <console port="920"/>
      <idle_pc>0x604f8104</idle_pc>
     </hw>
     <login password="cosas"/>
     <login user="ocsic" password="cafecafecafe"/>
     <enable password="passenable"/>
  </vm>

  <vm name="R2">
       <conf>/home/<user>/R2.conf</conf>
       <login password="cosas"/>
  </vm>
  <vm name="R3">
  </vm>
</vnx_dynamips>
```

Listado 23. XML de definición de un escenario expandido.

El fichero de configuración se descompone de dos partes diferenciadas. La primera parte, definida entre las etiquetas `<global>`, corresponde a parámetros de configuración que globales aplican a todos los *routers*. La segunda parte corresponde ya a la definición específica de cada *router*.

Todos los valores pueden ser definidos en la parte global o en la parte específica de cada *router* a excepción de tres: *sparsemem*, *ghostios* y *console_base*. Si hay valores definidos en la parte global y específica, se usarán los de la parte específica. Si no hay ningún valor definido o este fichero bien no existe o no está definido en el fichero de escenario del VNX, se utilizarán valores por defecto que a continuación de enumerará.

Explicación y valores por defecto de cada parámetro.

- *Sparsemem*. (true/false)

  Si es "true" reserva solo la cantidad de memoria necesaria para el sistema operativo de cisco IOS en vez de la definida en el XML del escenario. Permite crear más *routers*. El valor por defecto en caso de omisión es "true".

- *GhostIOS* (true/false)

  Si es "true" y si se va a utilizar la misma imagen para todos los *routers*, esta solo es llevada a RAM una sola vez y es utilizada por todas las instancias que utilicen esa imagen dejando otra porción de memoria independiente para la escritura. Con ello se consigue la reducción de utilización de RAM, ya que si es false, copiaría la misma imagen a memoria tantas veces como instancias habría. El valor por defecto en caso de omisión es "false" ya que al hacer pruebas se han encontrado con inconsistencias en los *routers* que incluso no llegaban a arrancar.

- *Enable*

  Con esta etiqueta se especifica la contraseña que se ha de insertar cuando se hace un *enable* en la terminal del cisco. La contraseña se especifica en el atributo *password* de esta etiqueta. Si no se especifica, no hay contraseña.

- *login*.

  Con esta etiqueta se puede especificar dos formas de autenticación en el *router*. Si solo se pone el atributo *password*, cada vez que se conecte al *router* a través de

alguna consola virtual, este le pedirá esta contraseña. Sin embargo si también se especifica usuario y contraseña, cuando se conecte al *router* a través de una consola virtual, este le pedirá que usuario y que contraseña va a utilizar para iniciar sesión. Cuando se especifican los dos tipos, prevalece el sistema de los usuarios dejando el primero inutilizado hasta que no se borren todos los usuarios del *router*. Por defecto no hay ningún usuario definido.

- *conf*

Esta etiqueta especifica el fichero de configuración con formato cisco que se le va a pasar al *router* como startup-config. Cualquier parámetro definido en el XML prevalece sobre lo que haya en este fichero.

Dentro de la etiqueta "*hw*" se definirán aquellos parámetros que sean de tipo hardware.

- *Console_base.*

Define el valor del primer puerto que se va a usar como entrada por telnet a los *routers* virtuales. Cada *router* creado tendrá un puerto asociado que será por el que se conectará un terminal para interactuar con el *router*. Por cada máquina virtual, se incrementara en uno el puerto a utilizar. Como puerto base en caso de que no se defina, se utilizará el 900.

- *Chassis*.

Especifica el chasis concreto que se va a utilizar dentro de una familia. Por defecto se utilizará el "c3640". Esta definición es importante ya que por ello van a depender que tarjetas se van a poder insertar y el número de ellas que puede soportar.

En la familia del 3600 hay los siguientes chasis con su correspondiente número máximo de tarjetas que puede admitir:

  - 3660: 6 Ranuras máximo
  - 3640: 4 Ranuras máximo
  - 3620: 2 Ranuras máximo

- *Slot*.

Define las tarjetas que se van a introducir en el chasis. El orden es importante ya que van a depender después el nombre de los interfaces.

En la familia del 3600 se pueden emular este tipo de tarjetas.

  - NM-1E (Ethernet, 1 Puerto)
  - NM-4E (Ethernet, 4 Puerto's)
  - NM-1FE-TX (Fast Ethernet, 1 Puerto)
  - NM-16ESW (Modulo Switch Ethernet , 16 Puerto's)
  - NM-4T (Serial, 4 Puerto's)

Por defecto se va a utilizar solo una tarjeta tipo NM-4E.

- *Console*

Este parámetro es por si se quiere especificar exactamente que la terminal de un *router* trabaje sobre un puerto específico. Si no existe se utilizará el valor por defecto que será el puerto base más el número de máquinas virtuales que haya iniciadas.

- *Idle_pc*

Cuando se emula un router, *dynamips* no puede traducir correctamente las instrucciones idle que manda el sistema operativo ya que estos sistemas operativos están hechos para diferentes procesadores, por lo que el resultado es que el proceso de *dynamips* ocupe normalmente el 100% del procesador. Para reducir la cantidad de procesador hay que pasarle un valor en hexadecimal al *dynamips* para que pueda reducir el consumo de CPU, este valor es el *idle_pc* y es diferente a cada procesador. La forma de calcularlo está en el Apéndice A. Si no se especifica nada se dará el valor 0x604f8104 que se ha comprobado que funciona en la mayoría de procesadores.

En caso de que no se especifique nada como el caso de R3 en el ejemplo o directamente no aparezca en este XML, se utilizaran siempre los valores por defecto.

4.6 Integración

Para hacer la integración de *dynamips* en VNX existen dos opciones diferentes:

1. Mediante la creación de ficheros de configuración para *dynagen* y que sea él el que realice toda la interacción con *dynamips*.
2. VNX se comunique directamente con *dynamips* para la creación y gestión de los *routers* que se van a virtualizar.

La ventaja de la primera opción es la sencillez que presenta la realización de un simple documento de texto donde se define el escenario y dejar al *dynagen* interactuar con *dynamips*, la desventaja es que por un lado es necesario utilizar otro software adicional y por otro es la imposibilidad de hacer tareas de gestión como podría ser la parada y rearranque de un solo *router*.

Estas desventajas hacen que no sea posible utilizar el software *dynagen* como intermediario dejándonos solo la posibilidad de que sea el propio VNX el que tenga que interactuar con *Dynamips* con la complejidad que eso conlleva, por lo que esta opción queda invalidada para la integración con el proyecto, por tanto, la elección que se llevó acabo es la de que sea VNX el que se comunique directamente con Dynamips para la gestión y ejecución de estos routers.

Como se dijo anteriormente, *Dynamips* detuvo su desarrollo en el 2007, sin embargo tampoco hay gran documentación sobre dicho hipervisor, por lo que la documentación que se puede encontrar por internet (incluso en el mismo paquete donde se encuentran los ficheros fuente de la última versión de *Dynamips*) resulta bastante antigua y muchas de las funciones que se especifican o han cambiado sus parámetros o han dejado de ser soportadas.

La única forma de poder saber los comandos que hay que enviar a *Dynamips* es capturando los mensajes intercambiados entre *dynagen* y *Dynamips* mediante *Wireshark*.

En la siguiente imagen se puede observar uno de los mensajes que envía *dynagen* a *Dynamips* para la gestión del routers, en concreto se está enviando el comando *vm set_ois R1* el cual le proporciona a *Dynamips* el fichero imagen de Cisco que se va a utilizar en ese escenario de prueba.

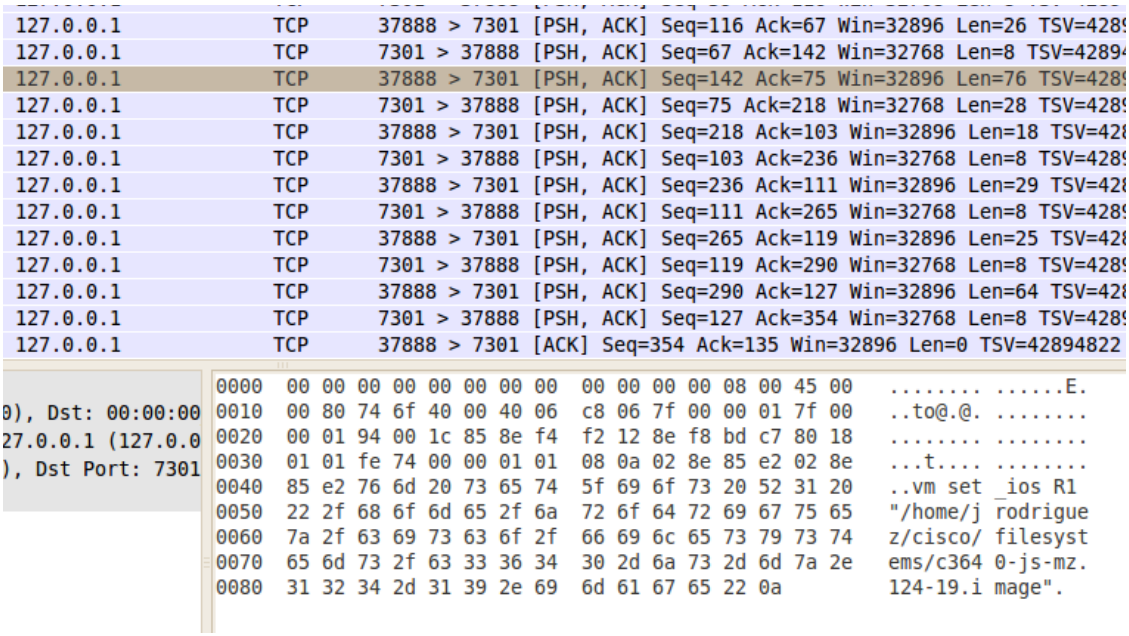


Fig. 16. *Wireshark* con un código de ejecución.

Con ello, se pudo identificar los comandos necesarios para poder realizar la ejecución y gestión de forma correcta de los routers virtualizados por parte de VNX en *Dynamips*. Estos comandos se pueden encontrar en el apartado de *Dynamips* en el estado del arte.

Con todos los comandos que se identificaron mediante la observación que se realizó por *Wireshark* en la ejecución de un escenario virtual creado a través de *Dynagen* y *GNS3* se pudo desarrollar las funciones necesarias para la gestión de los routers virtuales en VNX.

Para seguir con la filosofía del VNX, se creó un nuevo módulo llamado *vmAPI_dynamips* con la implementación específica de las llamadas genéricas:

Llamada	Detalles
defineVM	Define el router creando sus interfaces en el hipervisor y creando el fichero de configuración que se le va a insertar al router una vez que se ha arrancado. En esta función no se arrancará el router.
undefineVM	Elimina el router virtual del hipervisor mediante la llamada vm destroy.
destroyVM	Apaga el router virtual y elimina este en el hipervisor.
startVM	Inicia el router virtual mediante la llamada vm start.
shutdownVM	Apaga el router virtual.
saveVM	Extrae la configuración de los routers virtuales y la muestra.
restoreVM	Restablece los valores que en un primer momento se incluyeron en el fichero de configuración.
suspendVM	Suspende el router virtual en memoria.
resumeVM	Restablece el arranque del router virtual.
rebootVM	Mediante esta llamada se apaga el router virtual y se vuelve a arrancar desde el hipervisor.
resetVM	Similar a la llamada rebootVM, apaga el router y vuelve a arrancarse.
executeCMD	Debido a su extensión se tratará en un apartado separado.

4.6.1 Funciones especiales

4.6.1.1 ExecuteCMD

Mediante esta llamada se ejecuta un comando en el router virtual. Esta llamada es compleja ya que no se hace uso de ninguna de las primitivas que provee el hipervisor *Dynamips* en la gestión de los routers.

Para poder ejecutar comandos en el router virtual, en primer lugar se ha de conectar al puerto de gestión del router que se le haya asignado cuando se definió y arranco este. Una vez conectado al router, se ha de introducir el usuario y contraseña que se le estableció, adicionalmente se ha de controlar en que punto de se ha dejado en la última sesión abierta, esto se realiza mediante el paso del comando configure terminal a la consola del Cisco. Una vez que se ha realizado los pasos anteriores, se procede a enviar el comando ejecutable a router virtual.

La utilización es similar a una máquina virtual creada por libvirt en cuanto a creación, borrado o parada, pero hay un par de diferencias que por la forma que se ha diseñado *dynamips* no se pudieron salvar.

- La ejecución de comandos hay dos tipos.
    - *Verbatim*.

      Se ejecuta el comando en el router destino dentro del modo *enable*. Útil para ver estados con el comando *show* de cisco.
    - *File*.

      Si se quiere entrar dentro del mofo *config*, se ha de crear un fichero de texto con todos los comandos que hay que introducirle como si estuviera en una consola de ese router. Si se quiere entrar en el modo *enable*, también habría que poner ese comando en el fichero de configuración.

      Si en el modo *Verbatim*, el comando es un *reload*, no se mandaría al router, sino que se pararía ese router, se cambiar el fichero de configuración que tuviera por el especificado detrás del comando *reload* y se volvería a encender el router con la nueva configuración.

  En todos los casos hay una realimentación hacia el usuario de la respuesta que da el router cuando se ejecuta un comando.

No importa el estado en el que esté antes de mandar el comando, vmAPI_dynamips se encarga de dejar el router preparado en un estado conocido (fuera del *enable*) para poder realizar la ejecución del comando con seguridad.

4.6.1.2 destroyVM

Si dentro de una simulación hay varios routers iniciados y se quiere destruir uno y borrar sus interfaces, se tienen que destruir todos ya que se reinicia el *hypervisor*, esto se hace así para que el *hypervisor* deje libre los interfaces y puedan ser borrados adecuadamente por VNX. Con lo dicho anteriormente se recomienda no destruir routers siendo su solución temporal la creación de uno nuevo con interfaces diferentes.

Conclusiones

- Ellos, todos, dicen que estoy loco.
- No, claro que no estás loco, pero ¿qué pasaría si te dijera que estás soñando?
- No, no, no.
- ¿Y por qué no?
- Mire, yo sé lo que es real y esto es real.
- ¿Y tú como lo sabes? Los sueños no se descubren hasta que uno despierta.
- Lo sé y basta. Mis sueños son mucho más simples que todo esto.
- No, no hay ningún sueño simple. Mira a toda esta gente, parece que están hablando de sus cosas ¿a qué si?, completamente ajenos a ti, y sin embargo podrían estar ahí porque tú lo has querido, es más podrías hacer que se pusieran a tu servicio o, al contrario, que te destruyeran.
- Lo que quiero es que se callen y usted también.
(Todos se callan)
- ¿Lo ves?
- ¿Qué está pasando aquí? Que alguien me diga la verdad ¡joder!
- La verdad puede que no la soportaras.

Abre los ojos (1997)

5 Conclusiones

5.1 Resumen

En la realización de esta memoria se ha procedido en primer lugar a situar el proyecto dentro de un marco en el cual se centró en la posibilidad de utilización de VNX para la creación de *Honeynets*, definiendo así las razones que dieron lugar a la realización del proyecto y los objetivos marcados.

Adicionalmente se realizó un estudio de las principales tecnologías que actualmente existen dando ejemplo de productos comerciales o de desarrollo *opensource* que las implementan. En este apartado también se hizo hincapié en las tecnologías en las que se ha basado el proyecto (VNUML y VNX).

A continuación se explicó el desarrollo que se ha realizado para la integración de la solución de autoconfiguración y ejecución de comandos en Windows (VNXACE) habilitando así la posibilidad de ejecución de máquinas virtuales Windows mediante VNX. Seguidamente se realizó el estudio de la integración del software de virtualización *Dynamips* para dar la posibilidad de virtualizar routers Cisco.

Mediante los dos capítulos anteriores, se dio a VNX una capacidad de virtualización de diversos dispositivos que antes no estaban soportados. Gracias a eso, se posibilita que VNX sea una herramienta útil para la creación de *Honeynets* al poder crear una red heterogénea con una sola solución software.

Sin embargo, dicho proyecto no ha estado exento de problemas. En primer lugar, para poder dar soporte a diversas versiones de Windows (XP, Vista y 7) se necesitó dividir la solución en tres partes bien diferenciadas. La razón de esta división fue el poder capturar y ejecutar los comandos con diferentes perfiles de seguridad, los cuales cada uno te da una visibilidad limitada.

Otro problema fue el requisito necesario de no instalar software adicional en ninguna versión de Windows para ejecutar VNXACE para que, en el caso de descubrimiento de vulnerabilidades, no abrir una puerta de acceso a la máquina huésped. Este requisito invalidaba el desarrollo mediante C# ya que aunque de forma nativa está implementado en Windows 7, era necesario instalar software añadido en Windows XP.

Otro de los problemas que se han tenido en el desarrollo es debido a que *Libvirt* es un software en constante evolución, existieron versiones con errores los cuales no permitían realizar diversas tareas como por ejemplo, la imposibilidad de comunicación entre la máquina huésped y la virtual mediante puerto serie al no reconocer esta ningún tipo de puerto serie virtualizado aunque estuviese especificado, para solucionarlo se debió analizar el código de *Libvirt* y sustituir las líneas afectadas por líneas homologas de versiones anteriores.

El desarrollo de *Dynamips* tampoco ha estado exento de problemas. El principal dificultad fue la falta de documentación respecto a la API necesaria para poder utilizar este demonio. Esto complicó la integración de *Dynamips* en VNX ya que se necesitó realizar ingeniería inversa de las aplicaciones que utilizaban este demonio para estudiar el modo y la manera en la que se comunicaban.

5.2 Conceptos aprendidos

Gracias a este proyecto se ha permitido trabajar en un entorno empresarial ya que se estuvo colaborando estrechamente con la consultora S21SEC y Telefónica. Gracias a esta colaboración, se ha permitido desarrollar VNX con todos los comentarios y los puntos de mejora que nos remitían estas empresas, los cuales muchos de ellos no se habrían podido obtener en el grupo de desarrollo.

Adicionalmente se ha podido obtener una mayor experiencia en la gestión de grupos y trato con personas ya que el desarrollo de las soluciones anteriormente estudiadas en esta memoria se ha realizado en paralelo al crecimiento de la herramienta de VNX.

5.3 Trabajos futuros

Uno de los puntos a implementar en próximas versiones de VNX es el soporte de la definición de máquinas virtuales OVF haciendo que la herramienta permita leer este estándar y ser compatible con otros sistemas de virtualización.

Adicionalmente, también es interesante dar soporte a otro tipo de sistemas en red como pueden ser firewalls, ids, balanceadores o propios routers de otras marcas como pueden ser *Juniper* o *Teldat* que también están presentes en la mayoría de las redes de producción a las cuales nos intentamos asemejar.

Además, para evitar la duplicidad de funciones de la parte de VnxService y VnxDaemon en VNXACE, sería interesante estudiar la posibilidad de integración de estos ya que se reduciría el número de posibles fallos en la plataforma.

Uno de los inconvenientes de usar VNX es el gran consumo de CPU y memoria RAM que conlleva la virtualización de un escenario relativamente pequeño, por lo que es conveniente estudiar la posibilidad de poder realizar el despliegue de forma descentralizada, de forma que se distribuya parte del escenario en diferentes servidores y sean estos los que se encargan de su ejecución.

Otro de los proyectos a más corto plazo que se podrían realizar con esta base es la realización de *honeynets*, redes que son desplegadas de forma automática por un IDS (*Intruder detection system*) en el que se simule una red real. Para ello, esta herramienta tiene que ser integrada con un sistema inteligente el cual detecte la tipología de la amenaza y construya un escenario acorde al ataque que se está recibiendo

Otro de las posibilidades de dicho proyecto es la posibilidad de realizar pruebas de rendimiento, ya sean de aplicaciones cliente servidor o p2p, como mapas de red en el que se quieran realizar pruebas de carga o velocidad.

Bibliografía

You have to play the game, to find out why you're playing the game.

eXistenZ (1999)

6 Bibliografía

1. **Wikipedia.** Definición de virtualización. [En línea] http://es.wikipedia.org/wiki/Virtualizaci%C3%B3n.

2. VMware. TOP 100 Fortune. [En línea] http://www.vmware.com/es/technical-resources/advantages/customer/virtualization-customers.html.

3. *Green Computing.* [En línea] http://www.greenit-conferences.org/.

4. Blog Forrester. [En línea] http://blogs.forrester.com/category/virtualization.

5. **Madrid, Universidad Carlos 3 de.** Máquinas virtuales. [En línea] http://www.arcos.inf.uc3m.es/~folcina/pfc-html/node16.html.

6. **Cambrigde, Universidad de.** Página oficial del proyecto XEN. [En línea] http://www.cl.cam.ac.uk/research/srg/netos/xen/.

7. **Parallels.** Página oficial del producto comercial de Parallels. [En línea] http://www.parallels.com/es/products/virtuozzo/os/.

8. VNUML. [En línea] http://neweb.dit.upm.es/vnumlwiki/index.php/Main_Page.

9. Página oficial del proyecto de Libvirt. [En línea] http://www.libvirt.org.

10. *Sebek.* [En línea] https://projects.honeynet.org/sebek/.

11. Descripcion método registro de eventos. [En línea] http://technet.microsoft.com/es-es/query/aa363431.

12. *CreateNamedPipe.* [En línea] http://msdn.microsoft.com/en-us/library/windows/desktop/aa365150(v=vs.85).aspx.

13. Blog oficiales de desarrollo de Microsoft. [En línea] http://blogs.technet.com.

14. Página especializada de desarrollo de software Codeguru. [En línea] http://www.codeguru.com.

15. Microsoft Tech. [En línea] http://msdn.microsoft.com.

16. QEMU. [En línea] http://wiki.qemu.org/Main_Page.

17. Dynamips. [En línea] http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator.

18. Libvirt. [En línea] http://www.libvirt.org.

19. KVM. [En línea] http://www.linux-kvm.org/page/Main_Page.

20. Dynagen. [En línea] http://dynagen.org.

21. Codeproject. [En línea] http://www.codeproject.com.

22. Stackoverflow.com. [En línea] http://stackoverflow.com.

23. Página oficial de QEMU. [En línea] http://wiki.qemu.org/Main_Page.

24. Página oficial del proyecto KVM. [En línea] http://www.linux-kvm.org/page/Main_Page.

25. Página oficial del proyecto de Dynamips. [En línea] http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator.

26. Página oficioal del proyecto de dynagen. [En línea] http://dynagen.org.

27. Página oficial de desarrollo de Microsoft. [En línea] http://msdn.microsoft.com.

28. Páguna especializada de desarrollo de software stackoverflow.com. [En línea] http://stackoverflow.com.

29. Páguna especializada de desarrollo de software Codeproject. [En línea] http://www.codeproject.com.

30. Página oficial del proyecto VNUML. [En línea] http://neweb.dit.upm.es/vnumlwiki/index.php/Main_Page.

31. *User-mode Linux Kernel.* [En línea] http://user-mode-linux.sourceforge.net/.

Apéndices

I was drugged and left for dead in Mexico and all I got was this stupid t-shirt.

The Game (1997)

7 Apéndices

7.1 Apéndice A. Preparación de máquinas Windows.

7.1.1 Instalación.

Para la creación de una máquina virtual Windows se necesita el DVD del sistema operativo, en este caso se va a utilizar una imagen del DVD en ISO, por lo que los xml de creación estarán adecuados a la inserción de la imagen ISO en libvirt.

Primero hay que crear un disco duro virtual con el siguiente comando:

```
qemu-img create -f qcow2 root_fs_winxp-es-v01.qcow2 5GB
```

Siendo 5G la capacidad del disco duro virtual, en este caso son 5 Gigabytes. Es recomendable que la capacidad del disco cuando se quiera poner un Windows XP sea de 5 GB y en el caso de un Windows 7 sea de unos 20 a 30 GB.

Una vez que se ha creado el disco virtual, es hora de instalar el sistema operativo en él. Para ello se crea un XML (en este caso lo llamaremos root_fs_winxp.xml para WinXP) que va a ser el que se va a pasar al libvirt para crear la máquina virtual.

```
<domain type='kvm'>
  <name>WinXP</name>
  <memory>524288</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch="i686">hvm</type>
    <boot dev='hd'/>
    <boot dev='cdrom'/>
  </os>
  <features>
     <pae/>
     <acpi/>
     <apic/>
  </features>
  <clock sync="localtime"/>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <!--Se debe de modificar con la ruta absoluta donde esté el disco creado con el
comando anterior-->
      <source file='/usr/share/vnuml/filesystems/root_fs_winxp-es-v01.qcow2'/>
      <target dev='hda'/>
    </disk>
    <disk type='file' device='cdrom'>
      <!--Se debe de modificar con la ruta absoluta donde esté la imagen del sistema
operativo-->
      <source file='/almacen/iso/WINXP.iso'/>
      <target dev='hdb'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
    </interface>
    <interface type='network'>
      <source network='default'/>
    </interface>
    <interface type='network'>
      <source network='default'/>
```

```
    </interface>
    <interface type='network'>
      <source network='default'/>
    </interface>
    <!--graphics type='sdl'/-->
    <serial type='pty '>
        <source path='/dev/pts/3'/>
        <target port='0' />
    </serial>
    <graphics type='vnc' port='5005'/>
  </devices>
</domain>
```

Se crean con cuatro interfaces de red ya que cuando se inicie el Windows virtualizado se hace más rápido al no tener que cargar controladores nuevos.

Para iniciar la máquina virtual se ha de dar estos comandos:

```
virsh create root_fs_winxp.xml && virt-viewer WinXP
```

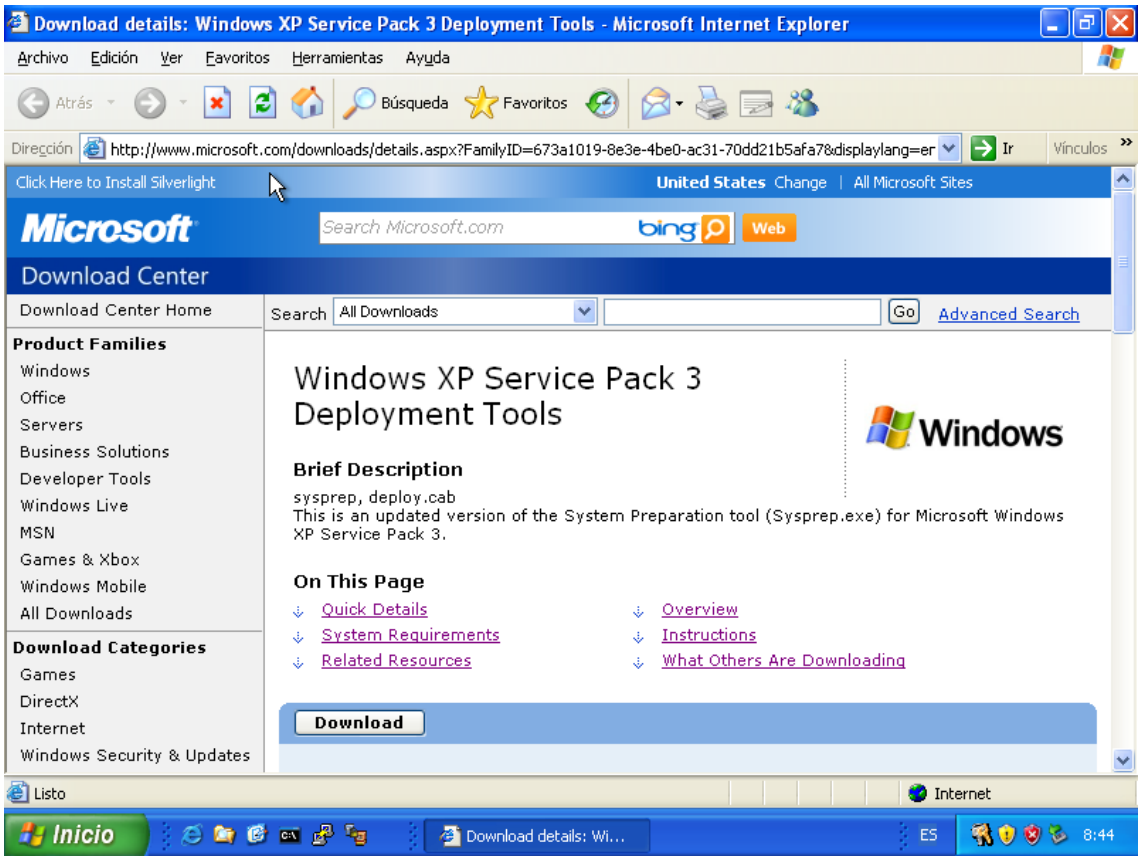
Y realizaremos la instalación del sistema operativo tal cual se haría en una maquina física.

7.1.2 Preparación con el sysprep.

Para máquinas Windows XP, en el caso de que no se active, procederá a dar 30 días de gracia permitiendo el uso total del sistema, una vez acabado ese periodo no dejará iniciar la máquina virtual. Por lo que si se van a utilizar sistemas Windows Xp sin ser activados durante mucho tiempo es recomendable seguir estos pasos para reiniciar el periodo de gracia ya que el periodo de gracia de las imágenes Windows XP utilizadas en un despliegue de un escenario mediante VNX no es reseteado y estarán contando a partir del día que fue instalada la imagen madre. Los sistemas Windows 7 no tienen ese problema ya que solo te informa que no está activado, pero no tiene periodo de gracia.

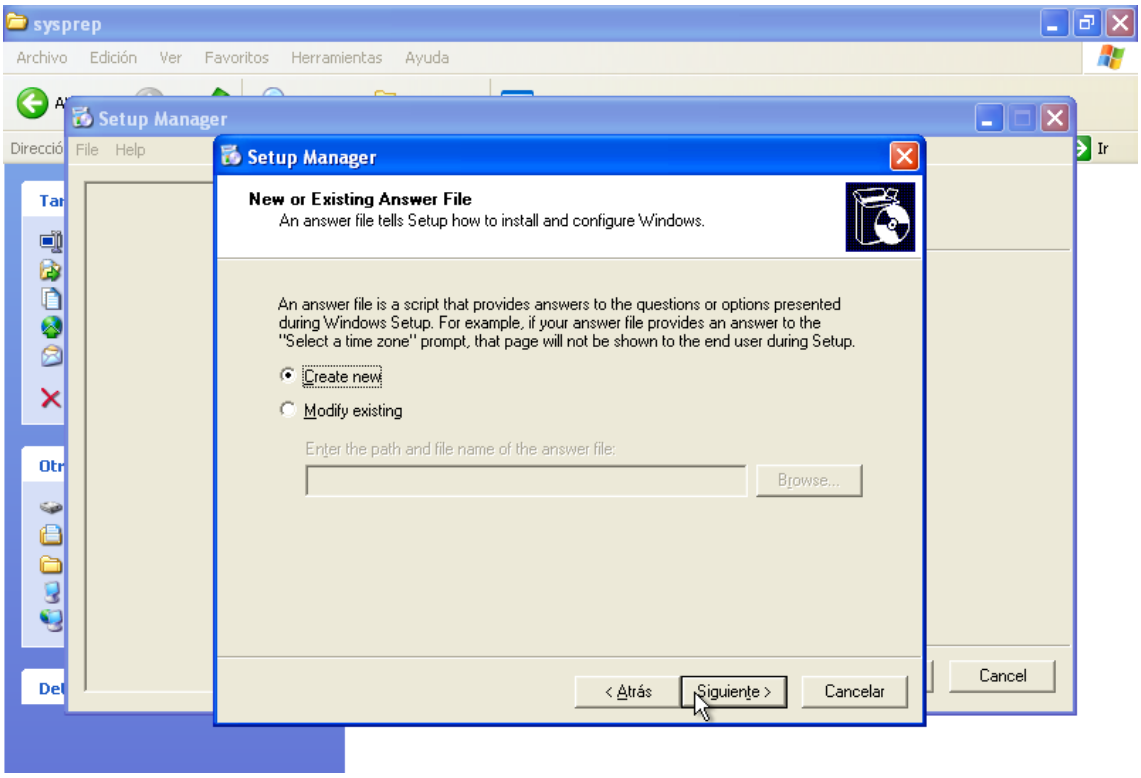
Instrucciones para el reseteo del periodo de gracia. Preparación de una máquina con sysprep:

1. Hacer una copia de seguridad de la imagen madre e iniciarla.
2. Bajarse el fichero deploy.cab disponible para el XP SP3 en esta dirección: http://www.microsoft.com/downloads/details.aspx?FamilyID=673a1019-8e3e-4be0-ac31-70dd21b5afa7&displaylang=en) o en el Cd de instalación que está en la ruta SUPPORT/TOOLS/Deploy.cab.

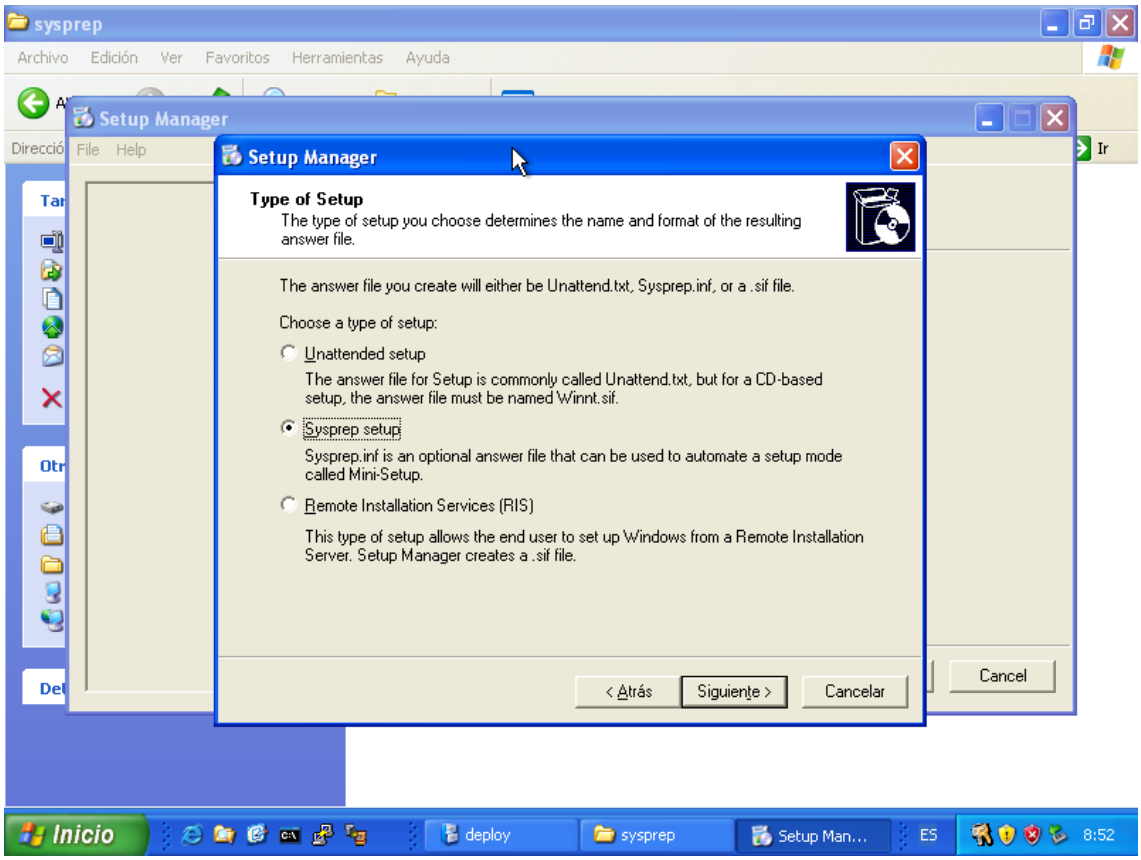


3. Descomprimir el contenido en la una carpeta nueva llamada sysprep dentro del directorio raíz de c.
4. Hay que crear el fichero sysprep.inf, para ello, se ejecuta la herramienta setupmgr que está dentro de la carpeta donde se descomprimió el fichero deploy.cab, esto es, c:\sysprep.
5. Seguir las instrucciones del agente de instalación. Una instalación recomendada sería la siguiente:

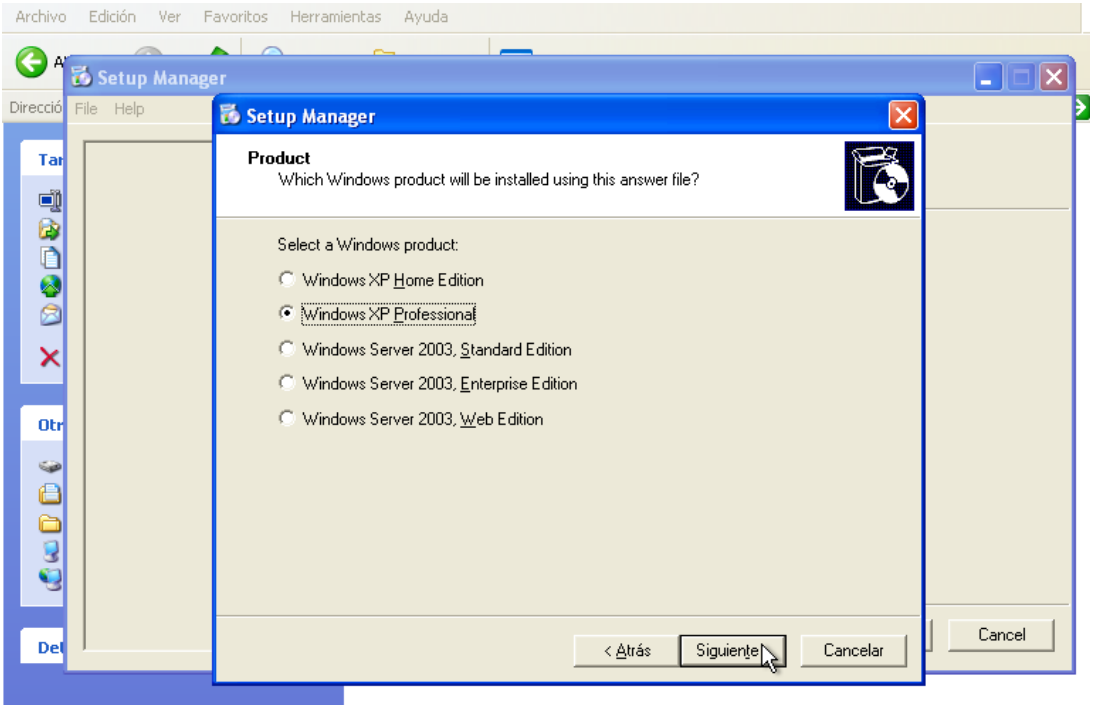
Crear un nuevo fichero sysprep.inf



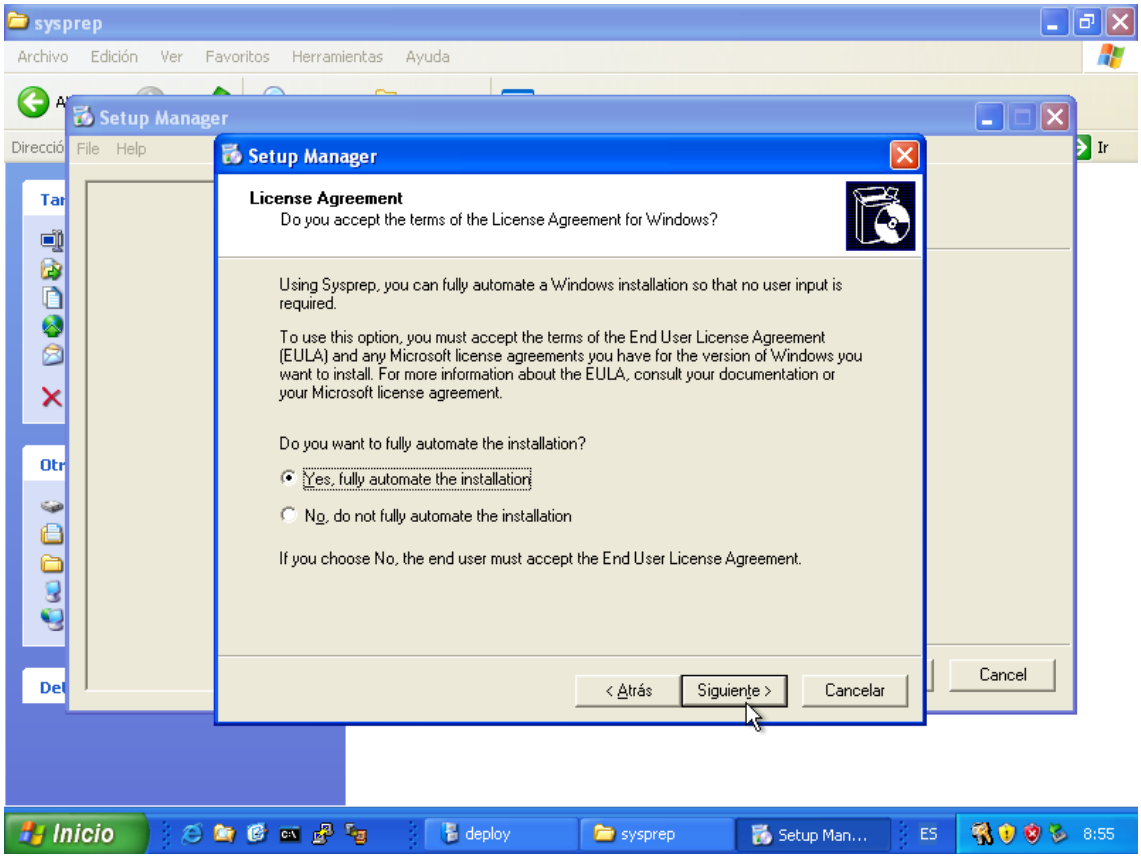
El tipo de fichero sea "Sysprep setup", ya que si eligiéramos la primera, se volvería a instalar el XP y pediría nombre de usuario, clave,...



Elegir el sistema operativo donde se va a instalar.

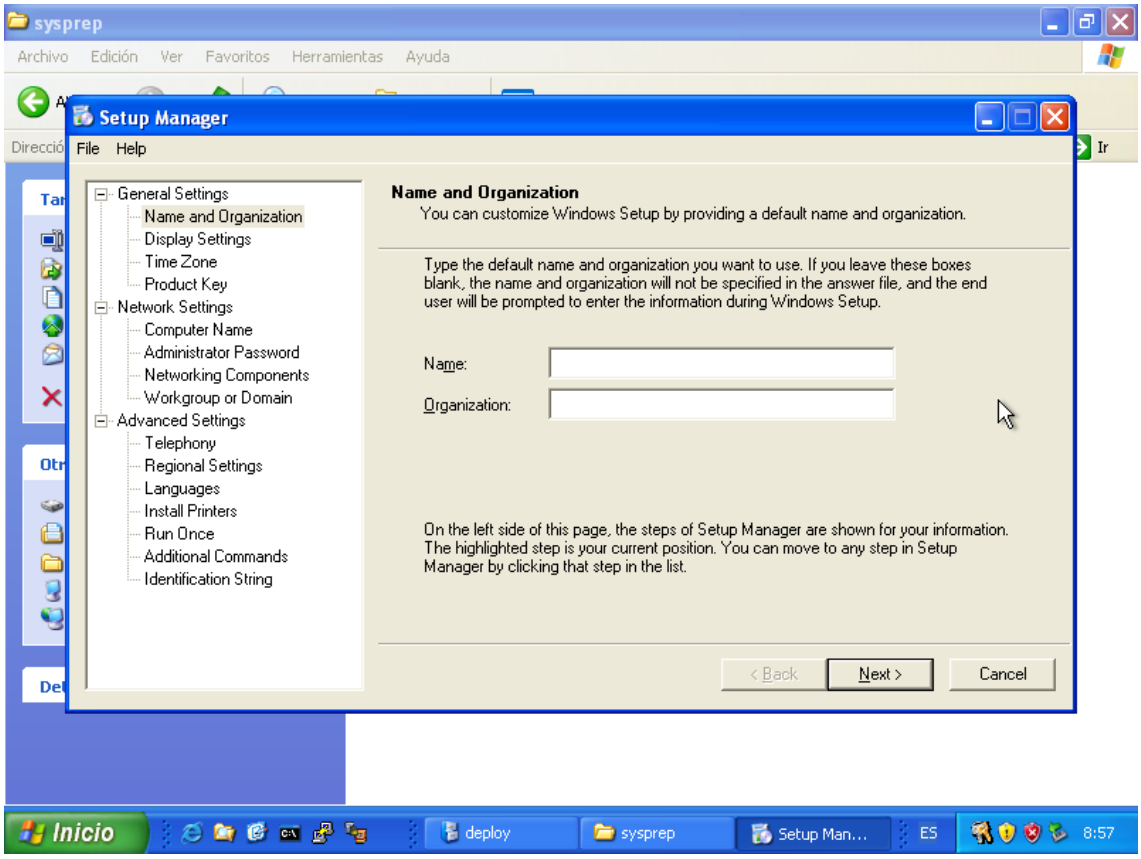


Que sea todo automático (más tarde elegiremos los parámetros de la configuración automática)

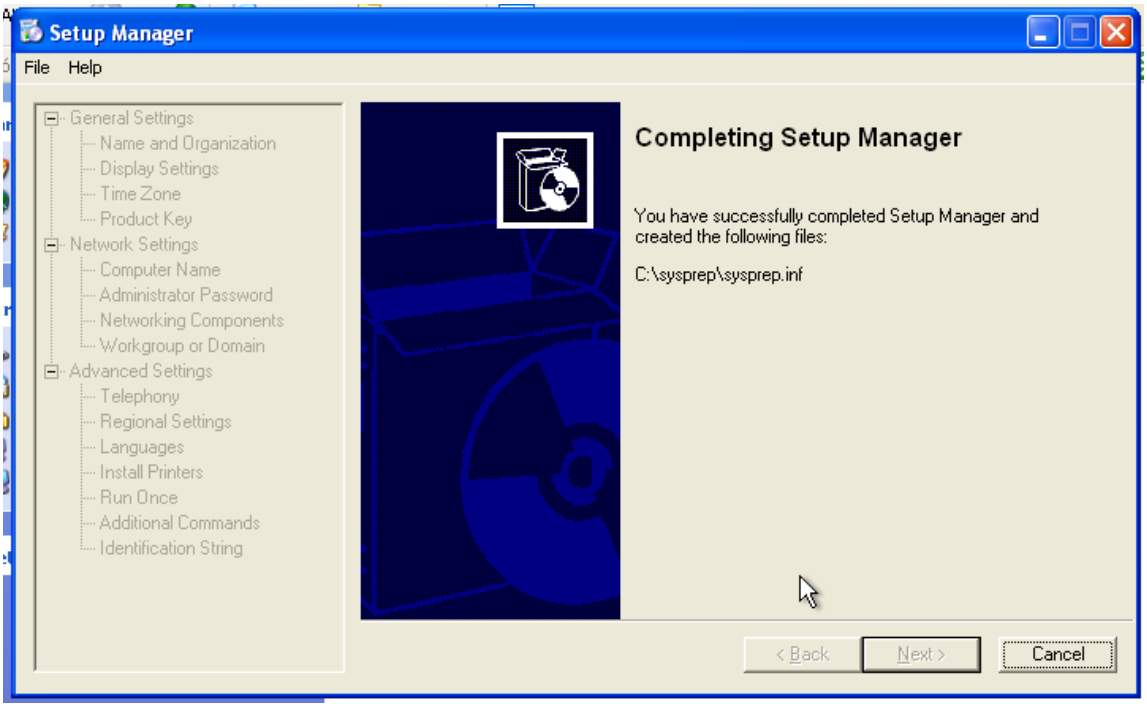


A partir de aquí elegiremos las diferentes configuraciones. Se recomienda ir una por una comprobando que todo esté OK, pero **evitar en todo caso pulsar sobre "Product Key" ya que se bloqueará la pantalla hasta que no le pongamos una clave valida.** Subproceso (obligatorio y recomendado):

1. En "Name and Organization" poner un nombre.
2. En Computer Name que se genere automáticamente.
3. En Administrator Password, introducir una contraseña de administrador y marcar la casilla de cifrado de la contraseña en el fichero sysprep.inf.

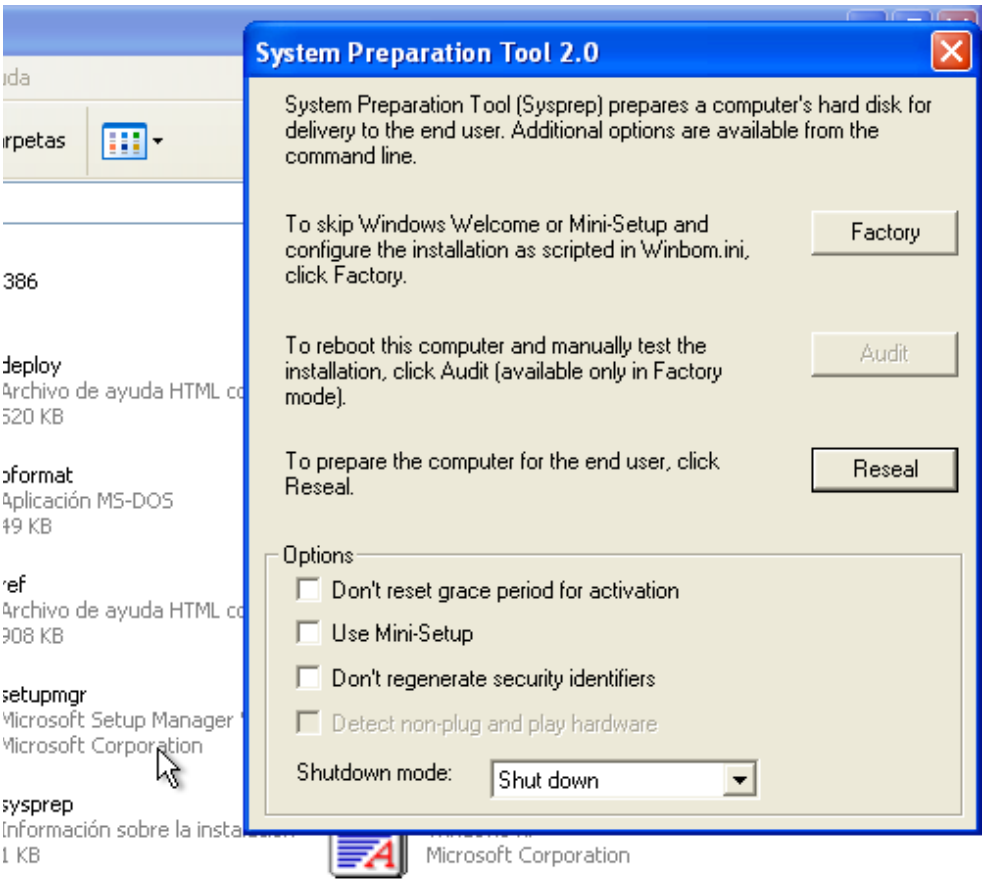


Le decimos que lo guarde en la ruta c:\sysprep\sysprep.inf (por defecto te pone esa).



Parece como si se hubiera quedado colgado pero no es así, solo hay que cerrarla, pulsando a la X, ya que se ha terminado el proceso.

6. Ejecutar sysprep.exe que está en el lugar donde se descomprimieron los ficheros. Le damos que aceptar a que nos cambie la seguridad del Windows.



Se elige en "Shutdown mode" el modo apagado y le damos al botón de "Factory", después se nos apagará la máquina. Ahora es el momento de realizar las copias que se quiera de la máquina virtual madre, Estas copias de momento no sirven para ser utilizadas como imágenes madres para ficheros qcow. Una vez realizada la copia, se inicia la máquina virtual de la imagen que se ha copiado. Una vez que se ha iniciado la máquina, se cierra la ventana que aparece de System Preparation Tool y se elimina el directorio c:\sysprep. Con todo esto la imagen copia estará lista para ser usada como fichero madre de las imágenes .qcow con el periodo de gracia reseteado a los 30 días.

Cuando finalicen esos 30 días, simplemente hay que hacer una copia de la imagen con el sysprep preparado y volver a quitar la carpeta sysprep.

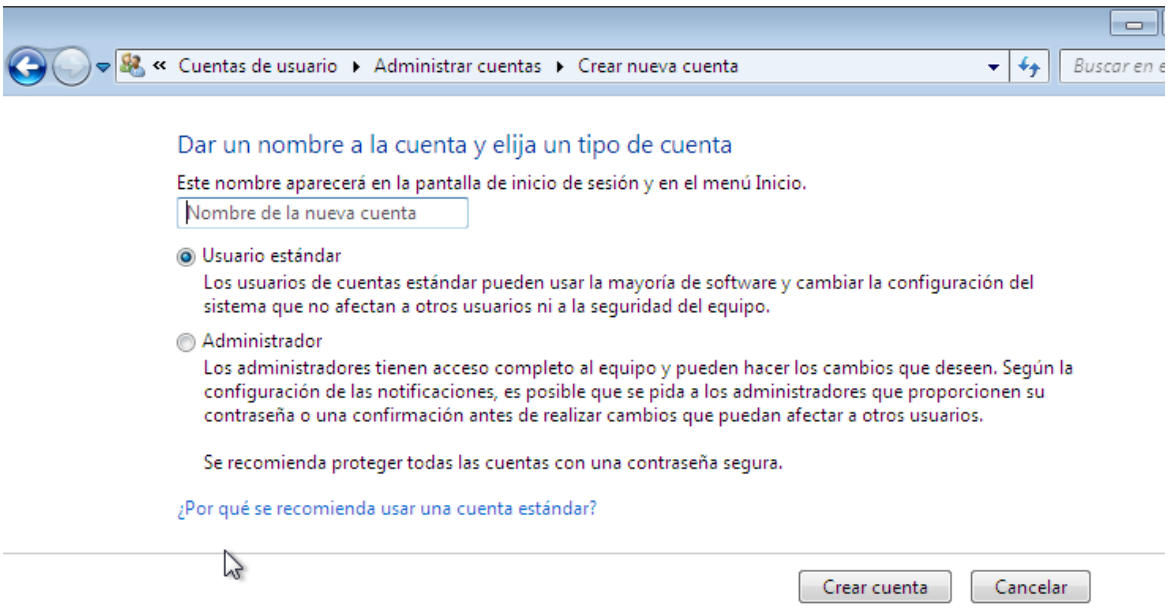
7.1.3 Creación de usuarios y autologin.

7.1.3.1 Creación de usuarios

En Windows 7, para la ejecución correcta del VnxClient, se ha de tener un usuario iniciado automáticamente, por lo que es altamente recomendable la creación de un usuario con muy pocos derechos y que se inicie de forma automática, también es recomendable hacerlo en las maquinas con Windows XP ya que después de la instalación se inicia un usuario con derechos de administrador de forma automática.

Para la creación de estos usuarios se hará un ejemplo con Windows 7.

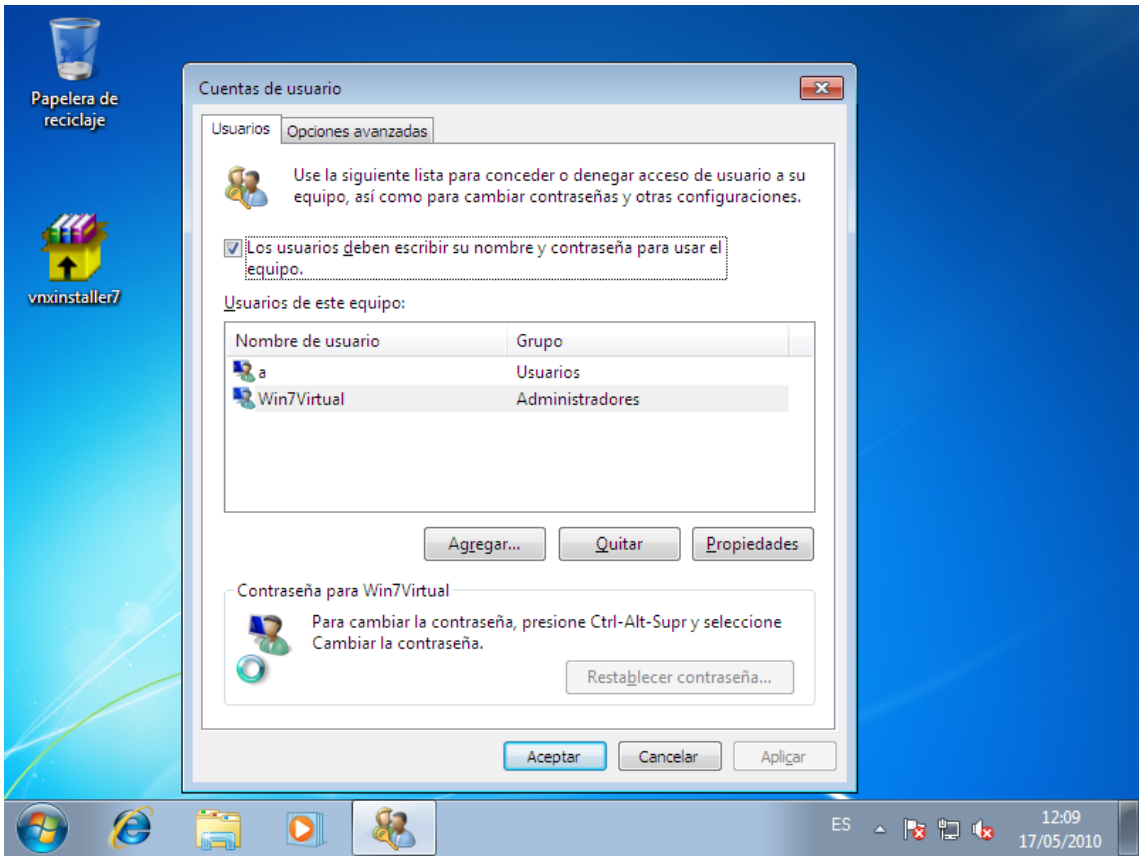
Se irá a Panel de control → Cuentas de usuario → Administra otra cuenta → Crear una nueva cuenta.



Se elegirá un nombre y se elegirá usuario estándar, opcional se puede poner password o no al nuevo usuario creado.

7.1.3.2 Autologin.

Para realizar el autologin también se hará mediante un Windows 7. Para ello se pulsará la tecla Windows + R y ejecutar `control userpasswords2`



Para activar el auto login:

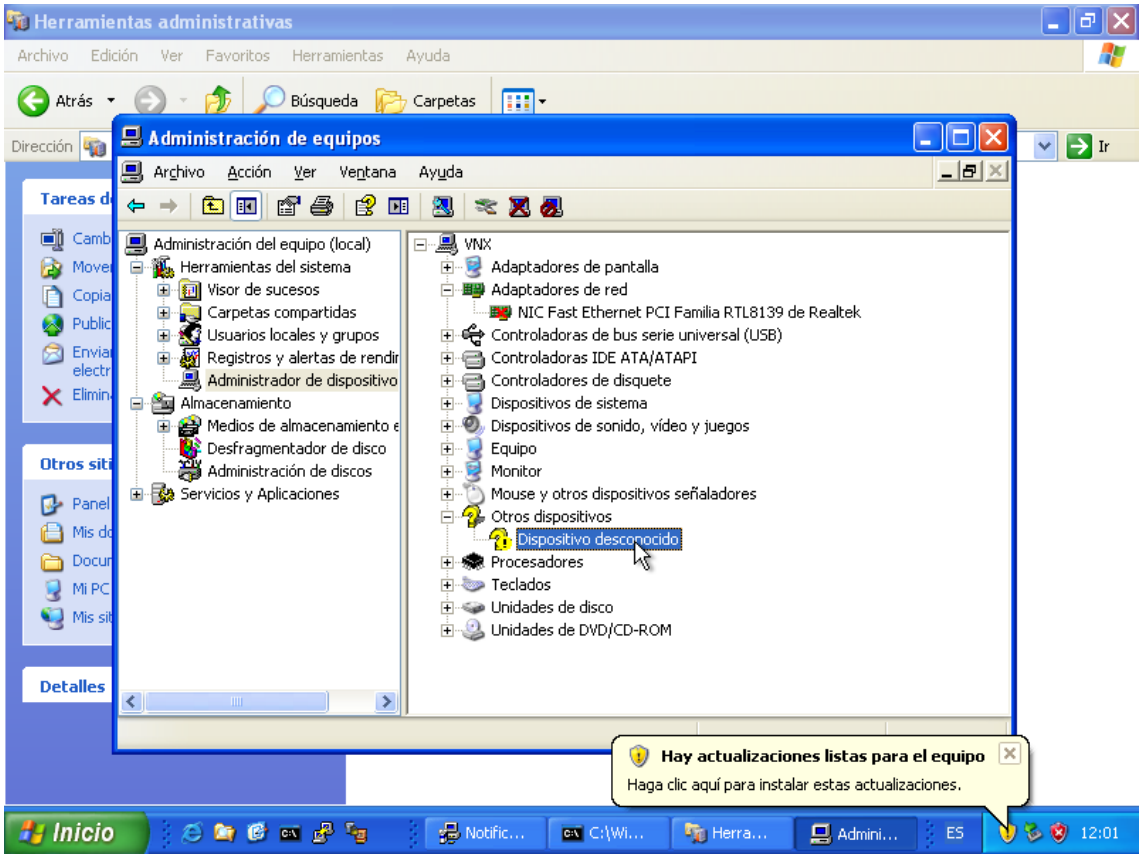
1. Seleccionar el usuario con bajos privilegios (en nuestro caso *a*).
2. Deseleccionar la casilla "Los usuarios deben escribir su nombre y contraseña para usar el equipo".
3. Le damos a aplicar.
4. En el caso de que se haya puesto contraseña a ese usuario, te la pedirá en este punto.
5. Le damos a Aceptar y ya tendríamos listo el autologin.

7.1.4 Finalización de la instalación.

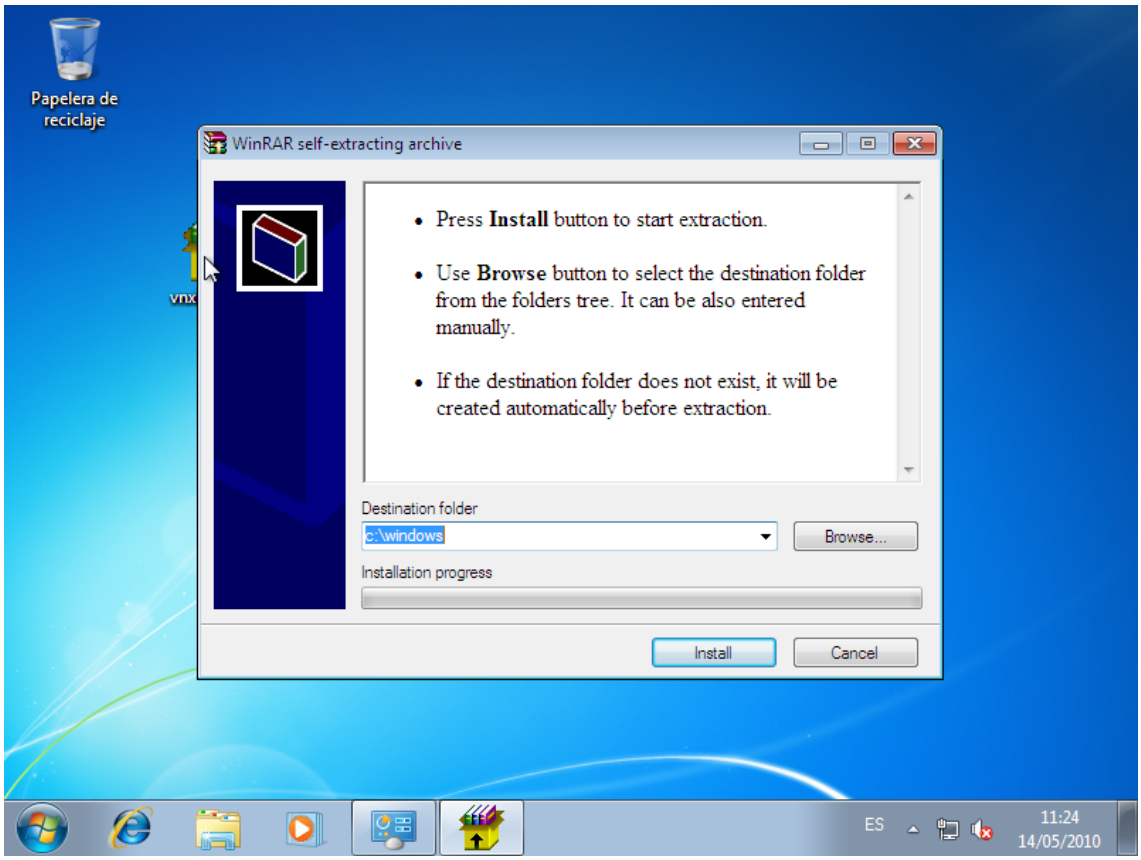
Cuando acabemos la instalación, preparamos la imagen con los programas esenciales que se querrán utilizar en las simulaciones teniendo en cuenta que cualquier cambio que se hagan en los escenarios virtuales lanzados con VNX no quedará guardado y que cada vez que se lance el escenario, se hará con un estado de la imagen tal cual la dejaremos en este punto. Si queremos modificar la imagen madre, se vuelve a iniciar la maquina con el comando anterior:

```
virsh create root_fs_winxp.xml && virt-viewer WinXP
```

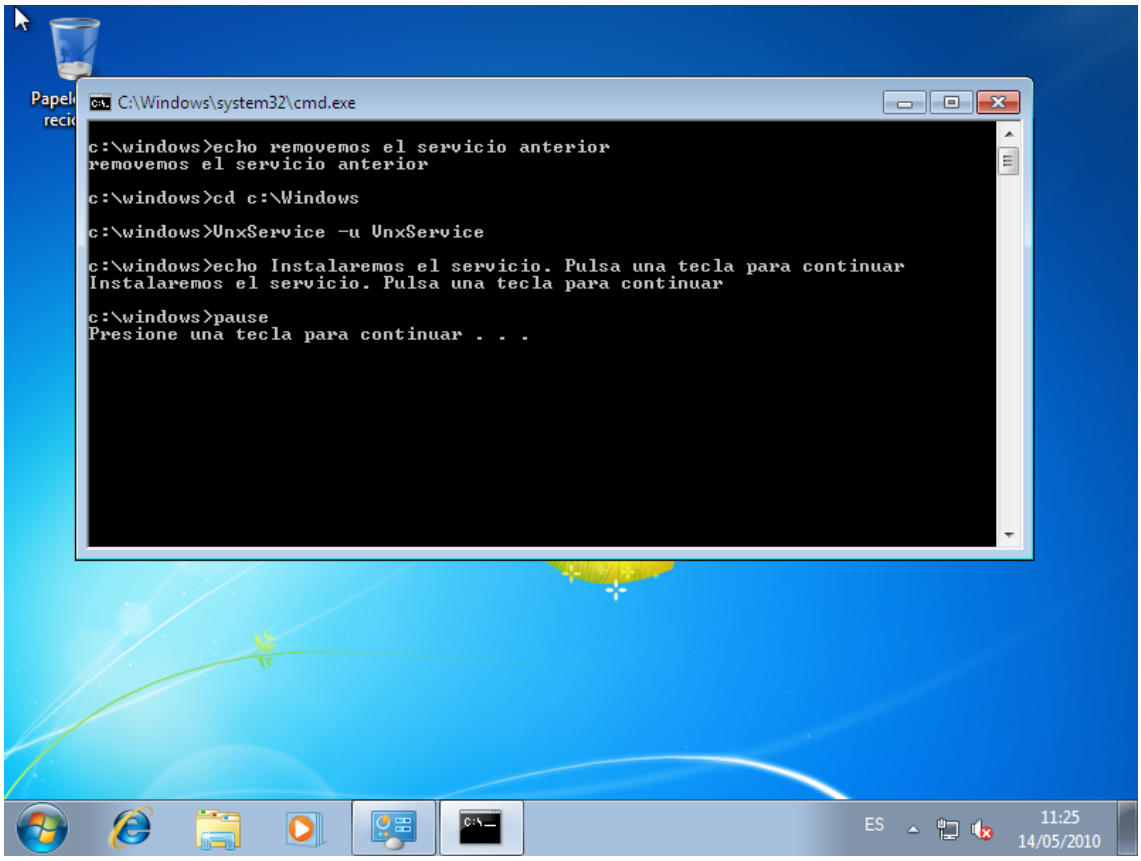
Antes de finalizar hay que asegurarse que se reconoce correctamente el puerto serie, si en el caso de que no se reconozca, hay que seleccionar el dispositivo desconocido y dar a actualizar controlador, a continuación que se conecte a Windows Update solo por esta vez, y por último a instalar automáticamente el software.



Lo último que haremos es instalar el demonio Vnxinstaller7.exe. Para ello se bajará el fichero y se ejecutará.



Los ficheros por defecto serán descomprimidos en c:\windows, además, también se instalará una ruta en el registro para que cada vez que se inicie una sesión se ejecute el VnxClient, la ruta donde se va a instalar la clave será la siguiente: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`



Por último para la creación y configuración de los entornos virtuales es conveniente que antes de apagar el sistema para dejarlo listo, se configuren todas las interfaces de red con IP estática y a continuación se desactiven, esto es así ya que hasta que no se completen todos los timeout de DHCP, no se procederá ni a cambiar el nombre del ordenador, ni a configurar la IP de los interfaces.

7.2 Apéndice B. Como calcular el idle_pc

Para calcular el idle_pc de un procesador, vamos a apoyarnos en la herramienta *dynagen*.

En primer lugar hay que definir en un fichero de texto un escenario simple en el que haya solo un router virtual, un ejemplo de este escenario puede ser el siguiente.

```
# simple1.net
[localhost:7300]
        [[7200]]
                image = /usr/share/vnx/filesystems/c3640
                # On Linux / Unix use forward slashes:
                # image = /opt/7200-images/c7200-jk9o3s-mz.124-7a.image
                npe = npe-400
                ram = 160
        [[ROUTER R1]]
                s1/0 = R2 s1/0
        [[ROUTER R2]]
                # No need to specify an adapter here, it is taken care of
                # by the interface specification under Router R1
```

Se ejecuta con `dynagen simple1.net` teniendo en cuenta que previamente se debe de haber ejecutado el hypervisor de *dynamips* también en el puerto 7300 de la siguiente manera `dynamips -H 7300`. Aparecerá la consola de dynagen por la cual se podrá dar órdenes.

Ahora mismo tenemos dos routers corriendo en nuestra máquina, pero para que las medidas que va a hacer dynagen sean correctas, hemos de dejar funcionando solo un router. Para ello, en la consola de dynagen, ejecutamos lo siguiente `stop R2`, con esto conseguimos parar el router R2.

Dejamos pasar unos minutos para que el router R1 configure sus interfaces de forma adecuada y se quede esperando a un comando.

Una vez que se ha configurado todo en el R1, volvemos a la consola de dynagen y ejecutamos el siguiente comando `idlepc get R1`. Como resultado te dará una lista de valores posibles de idle_pc, entre esos valores hay unos que están marcados con asterisco que esos son los que reducirán el uso de CPU. Dynagen elige el primer valor que aparece con asterisco y lo aplica automáticamente, por lo que se debería de notar un decremento en el uso de CPU. Si se quiere, se puede utilizar este valor para nuestro escenario o elegir entre los otros valores con asteriscos.

Algunas veces es posible que no aparezca ningún valor con asteriscos, por lo que tendríamos que dar otra vez el comando `idlepc get R1` pasado un tiempo.

7.3 Apéndice C. Listado de código.

7.3.1 Configuración del entorno.

```
varOut = docPtr->load(varXml);
     if ((bool)varOut == FALSE)
       throw(0);
     docPtr->get_firstChild(&pChild);
     while(pChild != NULL){
       pChild->get_childNodes( &List );
       List->get_length( &Amount );
       for( int i = 0; i < Amount; i++ )
       {
         List->get_Listado( i, &iNode );
         //Cambio de nombre del ordenador
         if(convert2(iNode->GetnodeName().GetBSTR())=="vm"){
           try{
                ComputerName=iNode->attributes->getNamedListado(_T("name"))->Gettext();
                TCHAR infoBuf[256];
                DWORD bufCharCount = 256;
                // Recogemos el nombre actual del ordenador
                if(!GetComputerName(infoBuf,&bufCharCount)){
                  trasmissiontx("Fail with computer name");
                  return 1;
                }
                wstring nombreordenadoract = infoBuf;
                CString maquina1 = nombreordenadoract.c_str();
                CString maquina2 = ComputerName;
                // Si no son iguales
                if(maquina1.CompareNoCase(maquina2)!=0){
                  // Se cambia el nombre del ordenador
                  if ((SetComputerNameEx(
                    ComputerNamePhysicalDnsHostname,ComputerName)==FALSE))
                  {
                    trasmissiontx(L"I can't change the computer name");
                  }
                  else{
                    trasmissiontx(L"Computer name changed. Reboot in progress.");
                    reinicio=1; // y se reinicia
                  }
                }else{
                  trasmissiontx(L"The computer won't be rebooted");
                  reinicio = 0; // Si los nombres son iguales, no se reinicia
                }
             }catch(...){
           }
         }
         // Fin del cambio de nombre del ordenador
         // Inicio de la configuracion de las tarjetas de red
         iNode->get_childNodes( &List2 );
         List2->get_length( &Amount2 );
         route = "";
         for( int i = 0; i < Amount2; i++ )
         {
           List2->get_Listado( i, &iNode2 );
           // Cambiamos el Route si está en el xml de la aplicacion
           if(convert2(iNode2->GetnodeName().GetBSTR())=="route"){
             route=iNode2->attributes->getNamedListado(_T("gw"))->Gettext();
             CString evento2="Route: ";
             evento2+=route.data();
             boolroute = 1;
           }
           //Activamos el Forwarding si está definido en el xml de autoconfiguracion
           if(convert2(iNode2->GetnodeName().GetBSTR())=="forwarding"){
             trasmissiontx("IP Forwarding enabled");
             ejecutar("sc config RemoteAccess start= auto");
             ejecutar("sc start RemoteAccess");
           }
         }
         List2->reset();
         EnableConnections();
         IEnumWbemClassObject* pEnumerator2 = NULL;
         hres = pSvc->ExecQuery(bstr_t("WQL"),bstr_t("SELECT * FROM
                  Win32_NetworkAdapter"),WBEM_FLAG_FORWARD_ONLY |
                  WBEM_FLAG_RETURN_IMMEDIATELY,NULL,&pEnumerator2);
         IWbemClassObject* pclsObj2;
```

```
  ULONG uReturn2 = 0;
  wstring macreconocida;
  // Configuracion de las tarjetas de red
  while (pEnumerator2)
  {
     hr = pEnumerator2->Next(WBEM_INFINITE, 1, &pclsObj2, &uReturn2);
     if(0 == uReturn2 || hr == S_FALSE)
     {
       break;
     }
     VARIANT vtProp3;
     VariantInit(&vtProp3);
     VARIANT vtProp2;
     VariantInit(&vtProp2);
     VARIANT vtProp4;
     VariantInit(&vtProp4);
     string comparacion;
     // Get the value of the Name property
     hr = pclsObj2->Get(L"ConfigManagerErrorCode",0,&vtProp3,0,0);
     hr = pclsObj2->Get(L"MACAddress", 0, &vtProp2, 0, 0);
     if (vtProp2.vt == 8 && vtProp3.intVal==0){
       macreconocida = vtProp2.bstrVal;
       comparacion=convert2(macreconocida);
       List2->reset();
       for( int i = 0; i < Amount2; i++ )
       {
          try{
           List2->get_Listado( i, &iNode2 );
           // Comparamos los datos de la MAC que se recogen en el XML
           // y los comparamos con los que recibimos del sistema
           if(convert2(iNode2->GetnodeName().GetBSTR())=="if"){
             ip= iNode2->Gettext();
             Mac=iNode2->attributes->getNamedListado(_T("mac"))->Gettext();
             Mac.erase(0,1);
             CString evento2="Mac: ";
             evento2+=Mac.data();
             CString interfaz1 = comparacion.c_str();
             CString interfaz2 = Mac.c_str();
             // Si son iguales, se procede al cambio de los parametros
             if(interfaz1.CompareNoCase(interfaz2)==0){
               CString evento2="IP: ";
               evento2+=ip.data();
               submask=iNode2->childNodes->GetListado(0)->attributes->
                        getNamedListado(_T("mask"))->Gettext();
               evento2="MASK: ";
               evento2+=submask.data();
               hr = pclsObj2->Get(L"NetConnectionID", 0, &vtProp2, 0, 0);
               string nombreinterfaz;
               if(vtProp2.vt > 1){
                if(comprobardatos(convert2(macreconocida),ip,submask)!=0){
                       nombreinterfaz=convert2(vtProp2.bstrVal);
                       // Si lo son, se procede a modificar ip y mascara
                       cambiarparametrosNIC(Mac,ip,submask);//,route);
                       if(boolroute == 1){
                              if(checkipmask(ip,route,submask)==0){
                                     cambiargateway(Mac, route);
                              }
                       }
                }
               }
               break;
             }
           }
          }catch(...){
           trasmissiontx("Mac error");
          }
       }

     }
     VariantClear(&vtProp2);
     VariantClear(&vtProp3);
  }

  pclsObj2->Release();
}
pNextSib = pChild;
pNextSib->get_nextSibling( &pChild );
```

```
}
// Al acabar de configurarlo todo, si se ha marcado que haya un reinicio
// se reinicia. Al volver a arrancar no hace falta volverlo a configurar todo
if (reinicio == 1){
  trasmissiontx(L"Disabling Network interfaces");
  DisableConnections();
  trasmissiontx(L"REBOOTING");
  ExecuteExternalFile(L"shutdown -r -f -t 0 ",false,false);
  //ejecutar("shutdown -r -f -t 0 ");
}else{
}
```

Listado 24. Código referente a la configuración del entorno

7.3.2 Espera de comandos.

```
int receptionrx()
{
    HANDLE hPiperx;
    LPTSTR lpvMessage=TEXT("Default message from client.");
    TCHAR  chBuf[BUFSIZE];
    BOOL   fSuccess = FALSE;
    DWORD  cbRead, cbToWrite, cbWritten, dwMode;
    LPTSTR lpszPipenamñe = TEXT("\\\\.\\pipe\\vnxletter");  // Nombre del pipe
    HANDLE hMutex;

    // Creamos el mutex para evitar
    hMutex = CreateMutex(
        NULL,
        FALSE,
        NULL);
    if (hMutex == NULL){
        trasmissiontx(L"Mutex hasn't been initialized");
    }
    while(true){
        // creamos el pipe
        hPiperx = CreateFile(
           lpszPipename,   // pipe name
           GENERIC_READ |  // read and write access
           GENERIC_WRITE,
           0,              // no sharing
           NULL,           // default security attributes
           OPEN_EXISTING,  // opens existing pipe
           0,              // default attributes
           NULL);          // no template file

        // Break if the pipe handle is valid.

        if (hPiperx != INVALID_HANDLE_VALUE) {
           dwMode = PIPE_READMODE_MESSAGE;
           fSuccess = SetNamedPipeHandleState(
               hPiperx,    // pipe handle
               &dwMode,  // new pipe mode
               NULL,     // don't set maximum bytes
               NULL);    // don't set maximum time
           if ( ! fSuccess)
           {
               return -1;
           }
           while (1)
           {
               do
               {
                    string letra = "D";
                    DWORD cbToRead =
(lstrlen((LPCWSTR)StringToWString(&letra).c_str())+1)*sizeof(TCHAR);
                    fSuccess = ReadFile(
                       hPiperx,    // pipe handle
                       chBuf,    // buffer to receive reply
                       BUFSIZE*sizeof(TCHAR),  // size of buffer
                       (LPDWORD) &cbToRead,

                       NULL);

                    if ( ! fSuccess && GetLastError() != ERROR_MORE_DATA )
                       break;
                    stringstream ss;
                    string s;
                    char c = chBuf[0];
                    ss << c;
                    ss >> s;
                    CString Listado = "CD Inserted -> ";
                    Listado.AppendChar(c);
                    Listado.Append(L":");
                    trasmissiontx(Listado);
                    DWORD dwWaitResult;
                    // Bloqueamos el mutex
                    dwWaitResult = WaitForSingleObject(hMutex,0);
                    switch(dwWaitResult){
                    case WAIT_OBJECT_0:
                       // Mandamos la letra al metodo insercioncd
```

```
                    // que se encargará de procesarla.
                    insercioncd(s);
                    ReleaseMutex(hMutex);
                default:
                    ;
                }
                Sleep(2000);
            } while ( ! fSuccess);  // repeat loop if ERROR_MORE_DATA

            if ( ! fSuccess)
            {
                trasmissiontx(L"ReadFile from pipe failed.");
            }
        }
    }
    Sleep(2000);
}
return 0;
}
```

Listado 25. Código referente a la espera y ejecución de comandos.

7.3.3 Copia.

```
void exefiletree(wstring entrada){
	wstring* origen = &entrada;
	// Escribimos un evento
	//lanzarevento("Error al ver el nombre del equipo","error");

	try{
		MSXML2::IXMLDOMDocumentPtr docPtr;
		MSXML2::IXMLDOMNodePtr DOMNodePtr;
		MSXML2::IXMLDOMNodeListPtr NodeListPtr;

		string scomando;

		wstring temp2;

		CoInitialize(NULL);
		docPtr.CreateInstance("Msxml2.DOMDocument.6.0");
		VARIANT vtTemp;

		vtTemp.vt=VT_I2;
		vtTemp.iVal = 1;
		if (origen == NULL){
			temp2= L"filetree.xml";
			origen = new wstring(temp2);
		}
		_variant_t varXml(origen->data());
		_variant_t varOut((bool)TRUE);
		varOut = docPtr->load(varXml);
		if ((bool)varOut == FALSE)
			throw(0);
		NodeListPtr = docPtr->getElementsByTagName("filetree");
		wstring wcomando;

		for (long i = 0; i<NodeListPtr->Getlength();i++){
		   long it = i+1;
		   string s;
		   stringstream out;
		   out << it;
		   s = out.str();
		   DOMNodePtr = NodeListPtr->GetListado(i);

		   MSXML2::IXMLDOMNamedNodeMapPtr AtrriPtr = DOMNodePtr->Getattributes();
		   wstring destino = AtrriPtr->getNamedListado(L"root")->
		                     Gettext().GetBSTR();
		   wcomando = L"xcopy ";
		   wcomando = wcomando.append((new wstring(1,entrada[0]))->c_str());
		   wcomando = wcomando.append(L":\\destination\\");
		   wcomando = wcomando.append(StringToWString(&s));
		   wcomando = wcomando.append(L"\\* ");
		   wcomando = wcomando.append(destino);
		   wcomando = wcomando.append(L" /H /O /X /C /E /I /Y");
		   scomando= convert2(wcomando);
		   CString temp = "Executing filetree command: ";
		   temp.Append(wcomando.c_str());
		   trasmissiontx(temp);

		   ExecuteExternalFile(scomando.c_str(),true,false);

		}
		senddataserial("Filetree finished\n");

	}catch( ... ){
		trasmissiontx("Read error of filetree");

	}
	//	CoUninitialize();
}
```

Listado 26. Proceso de copia de ficheros.

7.3.4 Ejecución de comandos

```
void execomando(wstring entrada){
	wstring* origen = &entrada;
	// Escribimos un evento
	//lanzarevento("Error al ver el nombre del equipo","error");


	try{
		MSXML2::IXMLDOMDocumentPtr docPtr;
		MSXML2::IXMLDOMNodePtr DOMNodePtr;
		MSXML2::IXMLDOMNodeListPtr NodeListPtr;
		MSXML2::IXMLDOMNamedNodeMapPtr AtrriPtr;
		string scomando;
		wstring modo;
		wstring gui;
		bool guib=true;
		wstring temp2;

		CoInitialize(NULL);
		docPtr.CreateInstance("Msxml2.DOMDocument.6.0");
		VARIANT vtTemp;

		vtTemp.vt=VT_I2;
		vtTemp.iVal = 1;
		if (origen == NULL){
			temp2= L"comandos.xml";
			origen = new wstring(temp2);
		}
		_variant_t varXml(origen->data());
		_variant_t varOut((bool)TRUE);
		varOut = docPtr->load(varXml);
		if ((bool)varOut == FALSE)
			throw(0);
		NodeListPtr = docPtr->getElementsByTagName("exec");
		int numprocesos=0;
		for (long i = 0; i<NodeListPtr->Getlength();i++){
			DOMNodePtr = NodeListPtr->GetListado(i);
			wstring wcomando = (DOMNodePtr->GetfirstChild()->text);
			AtrriPtr = DOMNodePtr->Getattributes();
			try{
					modo = AtrriPtr->getNamedListado(L"mode")->
						Gettext().GetBSTR();
			}catch (...)
			{
				modo = L"processn";
			}
			try{
				gui = AtrriPtr->getNamedListado(L"gui")->
				      Gettext().GetBSTR();
				if (gui.compare(L"yes")==0)
					guib=true;
				else
					guib=false;
			}catch(...){
				guib=true;
			}
			scomando= convert2(wcomando);
			CString temp = "Executing command in mode ";
			temp.Append(modo.c_str());
			temp.Append(L" : ");
			temp.Append(wcomando.c_str());
			trasmissiontx(temp);
			//lanzareventoE(temp.append(scomando),"informacion");
			//ejecutar(scomando);
			if (modo.compare(L"system")==0){
				ejecutar(scomando.c_str());
			}else if(modo.compare(L"processn")==0){
				ExecuteExternalFile(scomando.c_str(),false,guib);
			}else if(modo.compare(L"processy")==0){
				ExecuteExternalFile(scomando.c_str(),true,guib);
			}else{

			}
			numprocesos++;
		}
		string s;
```

```
                stringstream out;
                out << numprocesos;
                s = out.str();
                string mensaje = s;
                senddataserial(s.append("\n"));
        }catch( ... ){
                trasmissiontx(L"Executing command error") ;
        }
        //      CoUninitialize();
}
```

Listado 27. Extracto de código para la ejecución de comandos.

7.3.5 Envío de información a VnxDaemon desde VnxClient

```
#include "stdafx.h"
#include "Pipewrite.h"
#include "Util.h"
#include "GuiController.h"

#define BUFSIZE 1024
HANDLE hPipewrite;
int CreatePipeSend(VOID)
{
	BOOL   fConnected = FALSE;
	DWORD  dwThreadId = 0;
	LPTSTR lpszPipename = TEXT("\\\\.\\pipe\\vnxletter");

	// Creamos el Pipe
	hPipewrite = CreateNamedPipe(
		lpszPipename,             // pipe name
		PIPE_ACCESS_DUPLEX,       // read/write access
		PIPE_TYPE_MESSAGE |       // message type pipe
		PIPE_READMODE_MESSAGE |   // message-read mode
		PIPE_WAIT,                // blocking mode
		PIPE_UNLIMITED_INSTANCES, // max. instances
		BUFSIZE,                  // output buffer size
		BUFSIZE,                  // input buffer size
		0,                        // client time-out
		NULL);                    // default security attribute

	if (hPipewrite == INVALID_HANDLE_VALUE)
	{
		_tprintf(TEXT("CreateNamedPipe failed, GLE=%d.\n"),
GetLastError());
		return -1;
	}
	return 0;
}

int sendletter(CString lettercs){

	BOOL fSuccess = FALSE;
	BOOL   fConnected = FALSE;
	DWORD  dwThreadId = 0;
	DWORD cbBytesRead = 0, cbReplyBytes = 0, cbWritten = 0;
	// Conectamos con el pipe
	fConnected = ConnectNamedPipe(hPipewrite, NULL) ?
TRUE : (GetLastError() == ERROR_PIPE_CONNECTED);

	if (fConnected)
	{
		DWORD cbToWrite =
(lstrlen((LPCWSTR)(lettercs.GetString()))+1)*sizeof(TCHAR);
		// Escribimos la letra por el pipe
		fSuccess = WriteFile(
			hPipewrite,
			lettercs.GetString(),
			cbToWrite,
			(LPDWORD) &cbWritten,
			NULL);

	}
	return 1;

}
```

Listado 28. Envío de información a VnxDaemon desde VnxClient

7.3.6 Cambio de mascara a letra de unidad de CD

```
char FirstDriveFromMask (ULONG unitmask)
{
	char i;

	for (i = 0; i < 26; ++i)
	{
		if (unitmask & 0x1)
			break;
		unitmask = unitmask >> 1;
	}
	return (i + 'A');
}
```

Listado 29. Cambio de mascara a letra de una unidad de CD