UNIVERSIDAD DE ALCALÁ
Escuela Técnica Superior de Ingeniería Informática

INGENIERO EN INFORMÁTICA

Trabajo Fin de Carrera

"HERRAMIENTA PARA LA GENERACIÓN DE ESCENARIOS DE APOYO A LA DOCENCIA DE LA SEGURIDAD EN REDES"

Autor: Antonio Sánchez Camacho
Director: Iván Marsá Maestre

TRIBUNAL:

Presidente: Enrique de la Hoz de la Hoz

Vocal 1o: Dr. Bernardo Alarcos Alcázar

Vocal 2o: Dr. Iván Marsá Maestre

CALIFICACIÓN: _____

FECHA: _____

UNIVERSIDAD DE ALCALÁ
Escuela Técnica Superior de Ingeniería Informática

INGENIERO EN INFORMÁTICA

Trabajo Fin de Carrera

"HERRAMIENTA PARA LA GENERACIÓN DE ESCENARIOS DE APOYO A LA DOCENCIA DE LA SEGURIDAD EN REDES"

Antonio Sánchez Camacho
2011

A mi familia, mi madre, mi hermana y Lis, pero sobre todo a mi padre.
Os quiero.

6 “Herramienta para la generación de escenarios de apoyo a la docencia de la seguridad en redes”
Antonio Sánchez Camacho

Cuando empecé la carrera, una nueva etapa esperanzadora se abría ante mi, después de todos estos años, mucho esfuerzo, momentos buenos y menos buenos, se acaba. Me gustaría quedarme con las cosas positivas que han ido pasando por mi vida durante este tiempo, la gente que me ha apoyado y animado a seguir adelante, y a los que han recorrido el camino junto a mí.

Papá, Mamá, Elena, Lis, Iván, Adrián, Cristian, David, Adri, Morán, Álvaro, y un largo etc.

Una nueva etapa se abre, ni mejor ni peor, diferente.
Nuevos lugares, nueva gente, nuevos desafíos...

"Si yo no, quién?"

Índice de contenido

Índice de ilustraciones

16 “Herramienta para la generación de escenarios de apoyo a la docencia de la seguridad en redes”
Antonio Sánchez Camacho

Índice de tablas

RESUMEN

La seguridad de la información es un área de importancia creciente en la sociedad de la información. Sin embargo, las aproximaciones empleadas para su docencia son excesivamente estáticas y poco flexibles. Este Trabajo de Fin de Carrera establece unas bases sólidas para que los profesionales encargados de estas asignaturas, puedan dotar a los laboratorios de escenarios lo más realistas posibles, facilitando el despliegue de estos de manera automatizada y sencilla. Para ello, se ha desarrollado una herramienta abierta, distribuida, modular, escalable, extensible y basada en virtualización para la generación de escenarios de apoyo a la docencia de la seguridad.

1

INTRODUCCIÓN

El marco de trabajo de este proyecto se centra en intentar cubrir algunas lagunas en el área de los *Wargames* existentes, y su aprovechamiento para aumentar la calidad de los laboratorios que se imparten en asignaturas relacionadas con la Seguridad de la Información. Un *Wargame* suele consistir en una serie de pruebas con simulaciones de vulnerabilidades informáticas (pruebas de ingeniería inversa, seguridad de sistemas, criptografía, problemas de programación y algoritmia, etc.). Suele empezar con el rango más bajo que haya en el sistema, donde el usuario debe ser capaz de superar ciertas pruebas, a medida que las superes, tu rango ira aumentando y con ello la dificultad de las pruebas (Warzone , 2007).

En la actualidad, y echando un vistazo atrás, la mayoría de los *Wargames* carecen de un matiz fundamental: tener un cierto parecido con la realidad, de forma que el usuario que decide tomar parte en estos retos se encuentra casi siempre pruebas aleatorias, inconexas, con un parecido muy superfluo de un caso de uso real. Supongamos el escenario de una empresa ficticia, que ofrece ciertos servicios al exterior, los cuales se alojan en Servidores independientes (idealmente uno por cada servicio), que se encuentran dentro de una DMZ (*Demilitarized Zone*, Zona Desmilitarizada) cubiertos por un *firewall*, y que además poseen al menos una LAN interna, la cual se encuentra separada de la DMZ mediante otro *firewall*. Este escenario, aun siendo bastante básico, sirve para describir someramente lo que podría ser un escenario existente, y que en caso de poder implementarse en los laboratorios de las asignaturas relacionadas con la Seguridad de la Información podría servir para incrementar la comprensión de los alumnos desde un punto de vista tanto teórico como práctico de las debilidades existentes fuera del mundo académico, ayudándoles a practicar las técnicas usadas para la protección y la relización de auditorías de seguridad, formación que sin duda les sería de gran utilidad de cara al futuro.

Ya que nuestro propósito es que esta herramienta se pueda usar en los laboratorios de las asignaturas de Seguridad de la Información, debemos de intentar amoldarnos a los contenidos que se estudian en dichas asignaturas para tratar de que nuestros escenarios sean lo suficientemente configurables para poder cubrir el área de estudio de la disciplina. Como

base vamos a tomar los contenidos que se ofertan en la asignatura de Seguridad en Internet en la Universidad de Alcalá, por lo que nuestro *Wargame* deberá intentar contener un escenario en el que se pueda trabajar los siguientes contenidos (UAH., 2008):

- Introducción a la seguridad de sistemas.
- Analizadores de red.
- Acceso remoto.
- Vulnerabilidades locales.
- IDS.
- *Firewalls*.

Para abordar la tarea de desarrollar una herramienta que sea de utilidad en un ámbito docente altamente especializado debe tenerse en cuenta no sólo la funcionalidad de la aplicación, sino también las limitaciones de infrastructuras tanto físicas como lógicas inherentes al hecho de estar tratando con material muy específico y por lo tanto en ocasiones prohibitivo económicamente. Como medida paliativa se usará una tecnología en auge como es la Virtualización, esta tecnología tiene un enorme y esperanzador futuro (Pedro Tortosa, 2008).

Apoyándonos en el uso de la virtualización podremos hacer un uso eficiente del material existente en un aula de laboratorio, maximizando los recursos y proveyendo al usuario final, en este casos los alumnos, de una experiencia lo más realista posible. Mediante el uso de la Virtualización, en combinación con otras herramientas usadas en la administración de redes y sistemas, y basándonos en los escenarios definidos previamente por el personal docente, seremos capaces de desplegar a lo largo de un entorno de laboratorio una serie de máquinas virtuales, las cuales se comunicarán entre ellas tal y como se haya definido en el escenario anteriormente citado, por lo que los alumnos tendrán a su alcance y de manera casi real un escenario que podrán usar para conocer y practicar las técnicas usadas en el campo de la Seguridad de la Información.

Finalmente, vamos a hablar sobre el enfoque que se ha elegido a la hora de usar una u otra herramienta a lo largo de todo el proyecto. Se ha optado por hacer uso de la innumerable fuente de *software* libre y de código abierto disponible, como puede ser el alojado en *Sourceforge* y otros grandes repositorios existentes[1] . El *software* libre (GNU, 2009), es una cuestión de la libertad de los usuarios de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el *software*. El *software* de código abierto u *open source* (Open Source Initiative, 2011), es el término con el que se conoce al *software* distribuido y desarrollado libremente. El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código que a las cuestiones morales y/o filosóficas las cuales destacan en el llamado *software* libre. Esta elección está basada en un conjunto de ventajas que hacen que el uso de este tipo de *software* sea apropiado para el proyecto, entre estas ventajas encontramos: el *no coste* del *software* libre, por lo que no necesitaríamos de ninguna inversión económica. La posibilidad de *adaptar* y *personalizar* las herramientas elegidas a nuestro gusto, ya que podemos leer y modificar el código fuente de estas sin problema alguno. La *comodidad* a la hora de trabajar con este *software*, ya que llevo usando durante un largo periodo de tiempo herramientas de *software* libre, tanto a nivel de aplicación final, como de sistema operativo.

1 SourceForge (2011), http://sourceforge.net/

Una vez introducidos los componentes básicos del proyecto, se va a explicar la estructura seguida a la hora de desarrollar este Trabajo de Fin de Carrera.

En el capítulo 2 se hace una aproximación al *estado del arte*, donde se habla de los antecedentes históricos en cuanto al campo de la seguridad de la información se refiere, de la importancia y concienciación existente en este area, para pasar a hablar de la virtualización - tecnología imprescindible para el desarrollo del proyecto -, y finalmente dedicarle un apartado al *software* libre.

A continuación, en el capítulo 3, se pasa a hacer el *análisis del sistema*, capítulo en el que se estudiará detalladamente la propuesta realizada, para sacar un conjunto de requisitos que se usarán en capítulos posteriores. Se dividirá estos requisitos en 2 grandes grupos, funcionales y no funcionales, siendo estos últimos los que más tiempo nos ocupen, ya que se debe crear una serie de casos de uso y diagramas de actividad previos para poder entender y definir correctamente todos los requisitos.

Una vez que se han definido todos los requisitos, se pasa, en el capítulo 4, a realizar el *diseño del sistema* tomando a los anteriores como base. En este capítulo se realiza una división pormenorizada de las diferentes fases que componen lo que se puede llamar ciclo de vida de la aplicación desarrollada. Se empieza definiendo la arquitectura de las máquinas virtuales, para después analizar la manera de definir el entorno, desplegarlo, y gestionarlo. A continuación se dedica un apartado completo al estudio de la seguridad de la aplicación, y finalmente se diseña la aplicación desarrollada para este Trabajo de Fin de Carrera.

En el capítulo 5 se comienza a trabajar de manera práctica en el entorno de laboratorio, ya que se lleva a cabo la *implementación* del conjunto de herramientas usadas. En primer lugar se realiza la instalación del *software* necesario, para una vez hecho esto, pasar a configurarlo correctamente y conforme al diseño hecho en el capítulo anterior. En un último apartado, y de igual forma a lo hecho en el capítulo 4, se dedica exclusivamente a la seguridad, en concreto a poner en marcha las medidas discutidas previamente.

El siguiente capítulo, número 6, se dedica exclusivamente a la realización de una batería de *pruebas* que nos ayudarán a evaluar la madurez de la solución desarrollada, especialmente ante casos de error - ya sean voluntarios o involuntarios -, que con toda seguridad irán apareciendo cuando se despliegue en un entorno de producción y se haga uso del proyecto. En primer lugar se realizan las pruebas necesarias para verificar el funcionamiento en la fase de definición y parseo del escenario elegido por los profesores encargados del laboratorio. Una vez hecho esto, se continúa con las pruebas de despliegue y conectividad, para pasar a verificar el correcto funcionamiento de los servicios y eventos definidos, y en último lugar la gestión del escenario creado.

En el último capítulo del proyecto, se realiza una conclusión de todo lo visto anteriormente, resumiendo los puntos básicos e importantes de este Trabajo de Fin de Carrera, para finalizar hablando del trabajo futuro a realizar, que ayudará a mejorar y complementar lo hecho en este.

2

ANTECEDENTES

A lo largo de este capítulo vamos a introducir cuatro conceptos básicos, sobre los que posteriormente se apoyará el desarrollo del proyecto, enmarcando dichos conceptos dentro de las Tecnologías de la Información. Para poder ubicarnos, expondremos los *Antecedentes Históricos* en lo relativo al campo de la Seguridad, hablando de la evolución de los distintos tipos de vulnerabilidades y amenazas que han ido surgiendo durante estos años, y los daños que han ocasionado.

En segundo lugar, y ya introduciendo los conceptos a los que antes hicimos referencia, hablaremos sobre *La Importancia y la Concienciación sobre la Seguridad*, apartado donde se discutirá la situación actual en cuanto a la importancia que que le da a la Seguridad tanto el usuario medio como usuarios técnicos relacionados con las Tecnologías de la Información. Veremos que aunque cada vez la gente está más concienciada con la importancia extrema de mantener unas buenas políticas de Seguridad, aún falta mucho camino que recorrer, ya que hasta no hace mucho tiempo -incluso hoy en día- nos encontrábamos conque primaban otros factores en los entornos profesionales por delante de la Seguridad.

Para que el usuario medio tenga una buena concienciación en lo referente a los peligros y buenas prácticas en entornos tecnológicos, es imprescindible que los profesionales dedicados a la seguridad, que serán los encargados posteriormente de inculcar y transmitir la importancia de la seguridad, tengan unos conocimientos sólidos, los cuales adquirirán en su etapa formativa. Por ello, en el segundo apartado nos centraremos en los *Laboratorios de asignaturas de Seguridad de la Información,* la forma en la que ha evolucionado la docencia en este campo históricamente, y como encaja la propuesta que se hace en este Trabajo de Fin de Carrera para formar a futuros alumnos de estas asignaturas.

Uno de los factores más importantes a la hora de diseñar la herramienta desarrollada ha sido decidir la arquitectura que se va a usar, y dentro de esto como trabajar de una forma eficiente con los recursos *Hardware* existentes. Para maximizar estos recursos se hará uso de la tecnología de *Virtualización.* Discutiremos los diferentes tipos de virtualización que tenemos a nuestro alcance y el ámplio abanico de herramientas que nos permite usar esta tecnología, con los *pros* y *contras* de cada una.

Finalmente dedicaremos un apartado a explicar el porqué de la elección del *Software Libre* para el desarrollo del proyecto en todas sus fases, desde el Análisis y Diseño, pasando por la Implementación y las Pruebas, hasta la redacción de la Memoria final.

2.1. Antecedentes históricos

Corría el año 1972 cuando el primer virus de la historia, conocido como *Creeper*, se dio a conocer (Wikipedia, 2009 c). Este virus infectó una máquina IBM Serie 360, situada en ARPANET, la predecesora de lo que ahora conocemos como Internet. *Creeper* no era malicioso, en tanto en cuanto no dañaba los ordenadores infectados, sin embargo las máquinas víctimas del código malicioso emitían periódicamente en la pantalla el mensaje: *«I'm a creeper... catch me if you can!»* (¡Soy una enredadera... agárrame si puedes!). Aunque los efectos adversos en los ordenadores infectados eran nulos, surgió la necesidad de hacer frente a *Creeper*, creándose un segundo programa con el nombre de *Reaper*, aunque este programa fue diseñado para paliar los efectos de *Creeper* no podemos decir que es lo que hoy conocemos comunmente como antivirus, de hecho por su forma de actuar se aproximaría más a la definición de virus, ya que se autopropagaba por la red para desinfectar a las máquinas que habían sido infectadas con *Creeper*.

Hay muchos rumores alrededor de los autores de *Creeper* y de *Reaper*, de hecho se sospecha que el autor podría ser la misma persona. Las especulaciones en torno a la motivación que llevó a este individuo a realizar estas piezas de software son variopintas, desde que fue hecho por mera diversión, hasta que fue un experimento científico que se les fue de las manos, lo cual explicaría el hecho de que hasta el día de hoy el individuo que creó a *Creeper* ha permanecido en el anonimato.

Creeper fue el primero, pero muchos otros le han seguido desde entonces, al principio en ambientes controlados para poco a poco pasar a entornos más generalizados. El primer virus que operaba fuera de su entorno original de desarrollo fue el *Elk Cloner* (Virus Scan Software, 2011), creado por Rich Skrenta en el año 1981, con apenas quince años de edad. Afectaba a discos flexibles que se utilizaban en sistemas Apple II, y fue el primer virus en adoptar el modo de infección por sector de arranque. Cuando el ordenador se iniciaba con un disquete infectado, el virus se copiaba en la RAM e infectaba cada sector de inicio en cada nuevo disquete que se introdujera en la unidad, esparciéndose rápidamente. La inexistencia de un antivirus per se, hacía que el proceso de identificación y desinfección del *Elk Cloner* fuera bastante complicado. Para Skrenta no fue más que algo *«gracioso»* (Symantec, 2010), pero las infecciones mediante el sector de inicio serían adoptadas por muchos otros virus en los años siguientes.

Dos virus que tuvieron una atención especial durante la década de los ‘80 fueron el *Jerusalem* y el *Stoned*. Ambos compartieron el mismo año de nacimiento (1987), pero sus objetivos eran muy diferentes. En el caso del *Jerusalem*, el virus podía corromper todo archivo ejecutable en un sistema infectado cada Viernes 13 del calendario, e incluso podía afectar negativamente el rendimiento del ordenador. Para muchos, se trató del primer virus destructivo que operó a gran escala, frase que tomamos con pinzas teniendo en cuenta la cantidad de ordenadores activos en 1987, pero justificada debido a su enorme cantidad de variantes. En lo que se refiere al *Stoned*, su mensaje característico *«Your PC is now Stoned!»* (Su PC está apedreado!) todavía es muy recordado, al igual que su petición para legalizar la marihuana. El *Stoned* tuvo su amplia cuota de variantes, una de ellas cambiando el mensaje clásico por una referencia al 4 de junio de 1989, una de las fechas de la Masacre de la Plaza Tiananmen en China (Virus Wikia, 2010).

La década de los ‘90 fue testigo de un incremento importante en la cantidad de virus, pero nos vamos a quedar con dos que supieron dejar impronta en esos años. El *Michelangelo* fue todo un caso. La primera vez que se detectó fue en 1991 (IBM Research, 2009), y se esperaba que fuera el causante de una especie de *“día del juicio digital”*, borrando la información de millones de ordenadores alrededor del mundo. El virus se activaba el 6 de marzo, día del nacimiento de Miguel Ángel, sobreescribiendo sectores en disquetes y discos duros. Los medios realizaron una cobertura extensa, incluyendo por supuesto la inevitable pizca de paranoia que se le agrega a esta clase de noticias. Finalmente, los casos de pérdida de información fueron apenas una mínima fracción de los supuestos millones que vaticinaban los medios de comunicación, y el *Michelangelo* rápidamente pasó al olvido. En un aspecto mucho más preocupante, a mediados de 1998 hizo acto de presencia el virus *CIH* (Symantec, 2002), cuya primera versión se activaría el 26 de abril de 1999, coincidiendo con la fecha de la catástrofe nuclear en Chernobyl. En el peor de los casos, el CIH podía llenar el primer megabyte del disco de inicio con ceros, e incluso modificar la BIOS del ordenador, inutilizándolo por completo de forma efectiva. Si bien era posible restaurar un ordenador tras un ataque de este virus, no era un proceso sencillo.

Finalmente llegamos al siglo XXI, y con él llegó la *“Era del Caos”*, ya que la definición clásica de virus informático pasó a ser mucho más compleja de aplicar. El gusano *ILoveYou* dejó en evidencia lo eficiente que podía ser un e-mail para esparcir un virus. En 2003, Windows 2000 y Windows XP fueron puestos en jaque por la aparición del archifamoso gusano *Blaster*, que podía incluso apagar el ordenador sin intervención del usuario (Vsantivirus, 2003). Esto llevó a cambios profundos en los sistemas de Microsoft, y a una mayor concienciación por parte de la gente sobre la necesidad de tomar ciertas medidas mínimas de seguridad. En 2004 la masacre siguió con el *Sasser* y el *MyDoom*. *Zotob* marcó el inicio del “*malware lucrativo*” en 2005, y la introducción de las botnets como plataformas de malware.

En el primer mes de 2007, la botnet *Storm* era responsable del ocho por ciento del malware a nivel mundial en equipos con Sistema Operativo Windows (IEET, 2007). El funcionamiento de una botnet difiere ligeramente al de un gusano, en tanto en cuanto el segundo tiene un funcionamiento totalmente autónomo, mientras que las botnet pueden ser controladas por un único individuo, de forma que éste sería capaz de llegar a tener a su disposición miles o incluso millones de ordenadores, los cuales actuarán conforme a las instrucciones recibidas por este individuo. Un año después resulta imposible olvidarnos de *Conficker*, el cual generó pérdidas millonarias alrededor del mundo, y resultó ser un hueso extremadamente duro de roer. A día de hoy, estamos siendo testigos de lo que podríamos llamar *"guerra digital"*. El troyano conocido como *Stuxnet* tenía como blanco específico los sistemas de Control de Supervisión y Adquisición de Datos de Siemens, utilizados en territorio Iraní en desafío al embargo impuesto a ese país en 2006 debido a su programa de enriquecimiento de uranio. El 60% de las infecciones causadas por *Stuxnet* se localizaron en Irán, y provocó retrasos en la capacidad operativa de varias centrifugadoras de gas (alt1040.com, 2010).

Estamos a mediados del año 2011. ¿Que es lo que tienen preparados para nosotros los virus? Los expertos coinciden en que habrá una aceleración en las infecciones de malware sobre dispositivos móviles como tablets y teléfonos inteligentes, al mismo tiempo que buscarán aprovechar vulnerabilidades presentes en diferentes redes sociales (McAfee, 2011). Y el número seguirá creciendo. A principios de la década de los ‘90, los virus registrados no

excedían los mil cuatrocientos ejemplares. Hoy, tenemos más de doscientos millones de variantes, y se está haciendo cada vez más difícil deshacerse de ellas. Y todo, con aquel pequeño *Creeper* como punto de origen.

Existen, dentro del campo de la seguridad, un gran número de componentes *software* que ejercen efectos negativos sobre los ordenadores o los datos que se alojan en ellos, al conjunto de todos estos se les conoce hoy en día como *Malware* (InfoSpyware, 2009). El *Malware* es sin duda uno de los elementos dentro del área de la Seguridad de la Información más conocidos por el usuario medio, sin embargo, esta no es la única amenaza. Existen muchos tipos de vulnerabilidades y técnicas de explotación que no se valen del uso del *Malware* para llevar a cabo su cometido, mediante el uso de estas técnicas un usuario malintencionado puede ser capaz desde robar información personal, hasta aprovecharse del desconocimiento de los usuarios para obtener acceso a sus cuentas bancarias, pasando por por manejar a su gusto miles de ordenadores. Para conseguir estos propósitos, los ciberdelincuentes tratarán de aprovechar cualquier fallo, como ejemplo de esto podríamos basarnos en lo sucedido recientemente en lo que se conoce como *Operación Aurora* (El Economista, 2010), este caso nos servirá para introducir otros conceptos distintos al de *Malware*, pero usados igualmente por los ciberdelincuentes para perpetrar sus ataques.

A principios del 2010, varios empleados de Google situados en China y otros países, recibieron correos electrónicos *"extraños"*: los invitaba a acceder a una página de internet, a través de un link. Lo que siguió después ya se ha etiquetado como *"Uno de los ciberataques más sofisticados hasta ahora registrados"*, del cual fueron víctimas 35 empresas multinacionales, entre las que se encuentran la citada Google, Adobe, Rackspace,etc. El contenido de este correo no era más que un programa que se encargaba de aprovecharse de fallos de seguridad no reportados (*0-days*) para proveer *acceso remoto* a los atacantes. Este tipo de *software*, conocido como *exploit (Hacker, Anaya, 2008*), se vale de fallos lógicos a la hora de desarrollar el *software*, para hacer que este se comporte de una forma no esperada y ejecute una serie de acciones que ayudarán al atacante a llevar a cabo su objetivo. Una vez que los ciberdelincuentes obtuvieron *acceso remoto*, usaron otro tipo de técnica conocida como *escalada de privilegios*, la cual consiste en explotar un *bug,* fallo de diseño, o mala configuración en un sistema operativo o aplicación para poder acceder a recursos que normalmente están protegidos, como resultado de esto, podríamos llegar a ejecutar acciones que *a priori* no tendríamos acceso (School of Electronic Information and Electrical Engineering, 2008). Precisamente, esta fue la técnica usada por los atacantes para, una vez obtenido el acceso a la red interna de la empresas, llegar a conseguir documentos cuyo acceso estaba restringido a cierto grupo de usuarios.

Como hemos visto en este ejemplo acaecido recientemente, los ciberdelincuentes tienen a su disposición una amplia variedad de técnicas que van evolucionando en su grado de sofisticación y creciendo en número, las cuales les permiten llevar a cabo sus objetivos, Aunque los usuarios están cada vez más concienciado sobre la importancia de la seguridad, queda un largo camino por recorrer, sin duda las vías a seguir están estrechamente relacionadas con la concienciación y formación del usuario en una fase lo más temprana posible en su vida tecnológica, para que esto sea posible debe de haber profesionales lo suficientemente cualificados como para poder aleccionar de una forma didáctica pero metódica, una condición *sine qua non* para esto es que dichos profesionales tengan una base técnica sólida, tanto teórico como práctica, es en este escenario cuando entra en juego la herramienta desarrollada en el Trabajo de Fin de Carrera.

2.2. Importancia y Concienciación sobre la Seguridad

Si bien, desde los inicios de los ordenadores han existido riesgos sobre la seguridad, no ha sido hasta la creación y masificación de *Internet* cuando estos riesgos han alcanzado niveles inimaginables. *Internet* nos ha facilitado el acceso a todo tipo de información y servicios: entretenimiento, finanzas, comercio, etc. Quién iba a pensar hace unos años que íbamos a poder reservar una habitación de hotel al otro lado del océano, acceder a contenidos multimedia o consultar el saldo de nuestra cuenta bancaria a cualquier hora con un simple clic de ratón. Sin duda, esto ha supuesto un grandioso avance, que nos permite acceder a una enorme cantidad de funcionalidades desde la comodidad del sillón de nuestra casa, y no sólo eso, a nivel empresarial el desarrollo y explotación de los beneficios que nos aportan las nuevas tecnologías de la información es cada día mayor. Hoy en día sería impensable concebir una empresa que no tenga equipos informáticos, que no esté migrando, o ya lo haya hecho, la información de su negocio a formato electrónico, que no se introduzca en las nuevas formas de promocionar y vender sus productos que ofrece *Internet.*

Pero mientras que *Internet* es una herramienta casi indispensable en nuestra vida diaria, también es un reflejo de la sociedad actual en la que vivimos, con sus claros y oscuros. En este llamado *"lado oscuro"* nos encontramos a individuos que se aprovechan del desconocimiento del usuario medio para llevar a cabo todo tipo de actividades que repercuten negativamente en ellos. Además de los delincuentes cibernéticos, que intentaran poner más de una trampa con la que lucrarse a costa de los incautos, también hay una serie de riesgos vinculados al uso inadecuado de las tecnologías que se deben tener en cuenta.

En este apartado vamos a discutir sobre la necesidad de la concienciación en cuanto a la Seguridad, para conseguir esto analizaremos en primer lugar algunas de las *técnicas usadas* por los ciberdelincuentes y los *riesgos* que estas conllevan. Posteriormente nos centraremos en ofrecer *datos históricos* sobre las amenazas ocurridas en el pasado año 2010 y una *previsión de futuro* para este nuevo año, lo cual nos servirá para darnos cuenta de las repercusiones que tienen el no seguir una serie de directrices o buenas prácticas a la hora de securizar nuestro entorno y la velocidad con que evoluciona el área de la seguridad.

2.2.1. Técnicas usadas y Riesgos

Se acabaron los tiempos en los que el afán de los creadores de virus era el reconocimiento público. La evolución de las tecnologías de la información no ha pasado desapercibido para los delincuentes, que desarrollan en ellas sus actividades maliciosas con el objeto de lucrarse. Hoy en día es posible comprar un *keylogger* (herramienta que permite grabar en secreto todo lo que se escribe en el teclado del sistema intervenido, texto, contraseñas, números de cuentas,etc.) por 23 dólares, o pagar diez dólares por tener un ordenador conectado a Internet desde el que realizar *phishing* (posteriormente explicaremos este término). Esto es lo que dice un estudio de Symantec (Symantec, 2008), que concluye que la economía sumergida de este mercado negro está más que organizada, es autosuficiente y muy diversa geográficamente.

Entre el 1 de julio de 2007 y el 30 de junio de 2008, los investigadores de Symantec

estuvieron observando los *chats* y foros entre cibercriminales, y de la información extraída elaboraron todo un catálogo de códigos maliciosos y documentación detallada sobre el intercambio de datos financieros altamente valiosos. Por ejemplo, la información de tarjetas de crédito es la categoría más demandada, constituyendo más del 30 por ciento de todos los servicios vendidos en este mercado negro. Mientras que las credenciales de cuentas bancarias son las más publicitadas para vender con precios que van desde los 10 hasta los 1.000 dólares dependiendo del balance y localización de la cuenta.

Todo un negocio lucrativo, ya que de acuerdo con Symantec, si los vendedores pudieran vender todo lo que ofertan, la cantidad total sería de 275 millones de dólares, aunque contabilizando también las cuentas vacías de las víctimas y el saldo agotado de las tarjetas de crédito los números se sitúan ya en los 7.000 millones de dólares.

Por otro lado, este estudio también da a conocer las tendencias observadas en piratería de software durante el periodo de investigación. Los programas más pirateados son los juegos de escritorio seguidos de aplicaciones y software multimedia como editores de fotografías, animación en 3D y editores HTML. Además, geográficamente hablando, la mayor parte de las personas que descargan software pirata son de Estados Unidos, donde además, se encuentran la mayoría de los servidores utilizados para el mercado negro.

Otro aspecto interesante de estos métodos, y que también recalca el estudio, es que existe una conexión entre la piratería y lo que se vende por la calle. Es decir, un juego muy demandado y adquirido es igualmente el que más actividad ilegal genera.

En cuanto al perfil de los ciberdelicuentes, Symantec los divide en dos categorías: *individuos aislados* y *grupos sofisticados*. Unos y otros tienen como fin último obtener el mayor beneficio económico de cada una de sus acciones delictivas. Lo que explica que se haya llegado a constituir un auténtico mercado con una oferta y una demanda claramente definida. Estos individuos se valen de diferentes tipos de *Malware* que circula por la red en busca de ordenadores desprotegidos y usuarios incautos. Mediante el uso de este *Malware* llegarán a:

- Conseguir los datos bancarios o de servicios de pago en línea para suplantar nuestra identidad, así como cualquier tipo de información sensible.
- Utilizar nuestro sistema como pasarela para realizar otro tipo de actividades maliciosas: enviar correo basura, atacar otros sistemas, etc.

Las vías de entrada del *Malware* para infectar los sistemas son muy variadas y están en constante evolución. Las más comunes son:

- Al visitar páginas maliciosas, se aprovechan de agujeros de seguridad en el navegador y en los programas utilizados para ver las páginas: reproductores de vídeo, visores de texto (pdf), etc.
- Al abrir ficheros maliciosos, que llegan al sistema a través del correo electrónico, mensajería instantánea, redes P2P o descargados directamente de alguna página poco confiable.
- Al conectar al ordenador memorias USB que previamente han sido utilizadas en un PC infectado.

Otra de las técnicas principales que se están explotando con el uso masivo de *Internet* es lo que se conoce con el nombre de *Ingeniería Social*. Se define como la práctica de obtener información confidencial a través de la manipulación de usuarios legítimos. Es una técnica que pueden usar ciertas personas, tales como investigadores privados, criminales, o delincuentes cibernéticos, para obtener información, acceso o privilegios en sistemas de información que les permitan realizar algún acto que perjudique o exponga la persona u organismo comprometido a riesgo o abusos (Wikipedia, 2011 b). Del mismo modo que en la vida cotidiana cuando leemos una revista, vemos la televisión o hablamos directamente con una persona tenemos en cuenta una serie de factores que nos ayudan a poner en valor la información obtenida, en *Internet* debemos actuar de idéntica manera. Podemos clasificar las técnicas de Ingeniería Social principalmente en 2 casos de uso (ESET, 2009):

- Se incita al usuario a realizar algún tipo de acción *"necesaria"* para su navegación, cuando en realidad si el usuario sigue los consejos verá su sistema comprometido. Como ejemplo de este caso de uso podríamos pensar en una página que nos insta a descargarnos un complemento para nuestro navegador web que necesitamos para reproducir correctamente algún contenido de la página. Esta técnica fue la usada por el gusano *Sober*, el cual se fue propagando por correo con asuntos tales como "Re:Your Password" o "Re:Your email was blocked", alcanzando el récord de propagación en el año 2005 con cerca de una cota del 77% (Eroski Consumer, 2005).

- Se incita al usuario a introducir información privada con cualquier tipo de pretexto, momento en el que el atacante capturará dichos datos y los usara con fines fraudulentos. Como ejemplo de este caso de uso podemos hablas del *Scam* y el *Phishing*, técnicas en las cuales la víctima confía información privada creyendo que el destinatario de dicha información es una entidad legítima como puede ser un banco, cuando en realidad los datos irán a parar a un atacante.

Independientemente de la técnica utilizada, el modo de operar es siempre el mismo, hacer parecer que la información proveniente del atacante ha sido en realidad proporcionada por una entidad confiable, de modo que el usuario se sienta seguro y siga los pasos que se le indican y que en muchos casos le llevarán a ser infectado por diversos tipos de Malware. Para conseguir este propósito, los ciberdelincuentes usan los siguientes reclamos como medio de obtención de sus objetivos (ESET, 2009):

- Noticias sobre catástrofes.
- Famosos.
- Marcas y eventos conocidos.

Para llevar a cabo este tipo de ataques cada vez hace falta menos conocimientos, ya que hay herramientas que permiten el desarrollo de las técnicas anteriormente descritas sin muchos esfuerzos. Un ejemplo es el conocido *"Social Engineering Toolkit"* (SET, 2010), se trata de un conjunto de herramientas especialmente diseñadas para realizar ataques de Ingeniería Social en procesos de auditorías en seguridad, está programado en *Python* por David Kennedy. Este kit nos provee de un abanico enorme de ataques, algunos de estos son (Dragon Jar, 2010):

- *Phishing.* Nos permite generar automáticamente un sitio falso con el cual engañar a los destinatarios.

- *Vector de ataques Web. P*ermite realizar ataques automáticos a un usuarios que ingrese (por medio de ingeniería social) a una dirección que tu le especifiques (ahora lograr esto es mucho mas fácil gracias a los acortadores de direcciones), SET se encarga de subir el servidor y realizar el ataque que le especifiques (un *applet* de java, ataques múltiples, o *tabnabbing* entre otros) que te devolverá una *shell* en el equipo víctima.

- *Creación de medios infectados*. Permite crear un archivo que se conecte remotamente a nuestra maquina, ofreciéndonos una shell del sistema y se ejecute en el equipo remoto al introducirse una memoria/disco duro USB, o un disco DVD/CD aprovechando el "*autorun*" de windows.

- *Generación de ejecutables con payloads personalizados*. SET también permite (con la ayuda de *metasploit framework*) generar un ejecutable que se conecte remotamente a la máquina del atacante, una vez se abra en la máquina víctima, permitiéndonos configurar una gran cantidad de *shells*, entre ellas *meterpreter*, *shells* ciegas de windows, *shells* remotas,etc. Todo esto para procesadores de 32 y 64Bits.

- *Ataques por correo*. Se incluye una sección especialmente dedicada al correo electrónico, permitiendo enviar correos falsos o desde una cuenta *gmail*, a una o muchas personas.

- *Ataques con dispositivos personalizados Teensy.* Los dispositivos Teensy son todos aquellos aparatos en los que se ha utilizado la tableta programable Teensy en su elaboración, al ser altamente personalizable y programable se puede emplear en procesos de auditorías en seguridad (Irongeek, 2010), y SET nos facilita la tarea al programar estos dispositivos, ofrecernos rutinas que al conectar un dispositivo de estos nos podría poner a bajar una aplicación externa o ingresar a un sitio especifico y aceptar un applet de java, infectándonos en el proceso.

- *Falsificación de mensajes de texto (sms)*. Envío de *SMS* falsos, con los que podemos suplantar el numero telefónico que envía el mensaje, haciéndole pensar al receptor que efectivamente esa persona es quien le ha escrito. También incluye una opción para el envío masivo de *SMS*, por lo que podríamos enviar el mensaje a un mayor numero de destinatarios sin problema.

Como vemos, el potencial es enorme, y lo más sorprendente de todo es que no hacen falta grandes conocimientos técnicos para usar esta herramienta.
Los casos de Ingeniería Social son tan diversos como inabarcables. Los nombres de figuras o empresas conocidas y las noticias de importancia utilizadas para fraguar el engaño se actualizan constantemente, así como se renuevan los temas a los que se recurre para generar confianza en el usuario. El desconocimiento y la curiosidad son las vulnerabilidades que la Ingeniería Social explota. Por eso es importante que los usuarios se informen y eduquen. No todo aquello que es recibido por Internet, desde cualquier medio, es fidedigno y, si no fue solicitado, hay grandes posibilidades de que se trate de un malware o de un intento de engaño.

2.2.2. Datos estadísticos y previsiones de futuro

Las Tecnología de la Información evolucionan a una velocidad de vértigo, y el campo de la Seguridad de la Información no iba a ser menos, cada año surgen nuevas vulnerabilidades, nuevos vectores de ataque, nuevas amenazas que hacen que los profesionales tengan que estar alerta e informados constantemente bajo riesgo de quedarse obsoletos. Vamos a dividir esta sección en 2 apartados, con el primero de ellos obtendremos una visión general de las principales amenazas que azotaron al campo de la Seguridad de la Información durante el pasado año 2010, y en el segundo de ellos echaremos un ojo a las predicciones de amenazas que se esperan a corto plazo.

2.2.2.1. Principales datos del 2010

Vamos a basarnos en el informe realizado por *Web Sense* (Web Sense, 2011) el cual resume los descubrimientos significativos de los investigadores de *Web Sense* mediante el uso de la Red *ThreatSeeker* durante 2010. La Red *ThreatSeeker* escanea más de 40 millones de sitios Web por hora en busca de código malicioso, y casi 10 millones de *emails* en busca de contenido no deseado y código malicioso.
Daremos un vistazo a las principales amenazas que surgieron en el año 2010 relacionadas con incidentes de Seguridad. Para comenzar, observemos en la *tablas 1, 2 y 3* algunas estadísticas clave obtenidas.

Seguridad Web
Se identificó un crecimiento de un 111.4% de sitios web maliciosos con respecto al año 2009
El 79.9% de los sitios web con código malicioso resultaron ser sitios web legítimos que habían sido comprometidos. El incremento con respecto al periodo anterior fue de un 3%
Los Estados Unidos fue el país que hospedo mayor número de páginas web relacionadas con técnicas de phishing

Tabla 1: Seguridad Web

Seguridad Correo Electrónico
El 84.3% de los correos fueron spam. Hubo un decremento del 0.7% con respecto al periodo anterior
El 89.9% de los correos no deseados en circulación durante este periodo contenía enlaces a spam o sitios web maliciosos. Se produjo un incremento de un 4% con respecto al periodo anterior
Las compras permanecieron como primer tema en el spam

Tabla 2: Seguridad Correo Electrónico

Seguridad de Datos
El 9% de los robos de datos ocurrieron vía correo electrónico
El 52% de los robos de datos ocurrieron vía Web
Los Estados Unidos fue el primer país donde se produjeron conexiones de Malware
Los Estados Unidos y China continuaron en el top 2 de países donde se alojaron organizaciones cibercriminales y recibieron datos robados durante el 2010. Holanda pasó a formar parte del top 5 de estas estadísticas.

Tabla 3: Seguridad de Datos

En el 2010 hubo muchos hechos destacados en cuanto a la (In)Seguridad Informática se refiere, pasamos a comentar brevemente los más destacados:

- El gusano *Stuxnet* da un paso más y ataca sistemas SCADA de Alto Riesgo que controlan procesos industriales.
- Arrestado un joven de 20 años que con ayuda del troyano *Zeus* robó más de 600 datos de cuentas bancarias. Apropiándose de unos 3 millones de dólares por mes hasta un total aproximado de 20 millones de dólares antes de ser detenido.
- Operación Aurora, ataque a gran escala utilizando una vulnerabilidad *0-day* en Internet Explorer (Microsoft, 2010). Entre las víctimas encontramos el nombre de grandes empresas como Adobe, Google, Rackspace, Northrop Grumman, Dow Chemical y los gobiernos de Alemania y Francia.
- Ataques a WordPress, la plataforma de *blogging* con mayor uso en el mundo entero.
- El Hospital de Massachusetts pierde 800.000 ficheros de datos con información personal recopilada a lo largo de 15 años.
- *Hackers* sacan a la luz más de 100.000 cuentas de usuarios de AT&T que usaban el iPad 3G.

A continuación vamos a pasar a comentar una serie de gráficos que nos permitirán sacar conclusiones importantes. En la *ilustración 1* podemos ver las estadísticas del número total de páginas visitadas que contenían *links* maliciosos. Como se observa, las páginas con contenido sexual están a la cabeza, seguidas de *webs* con contenido tecnológico y de noticias, esto va en consonancia al tipo de contenidos que consume el usuario.

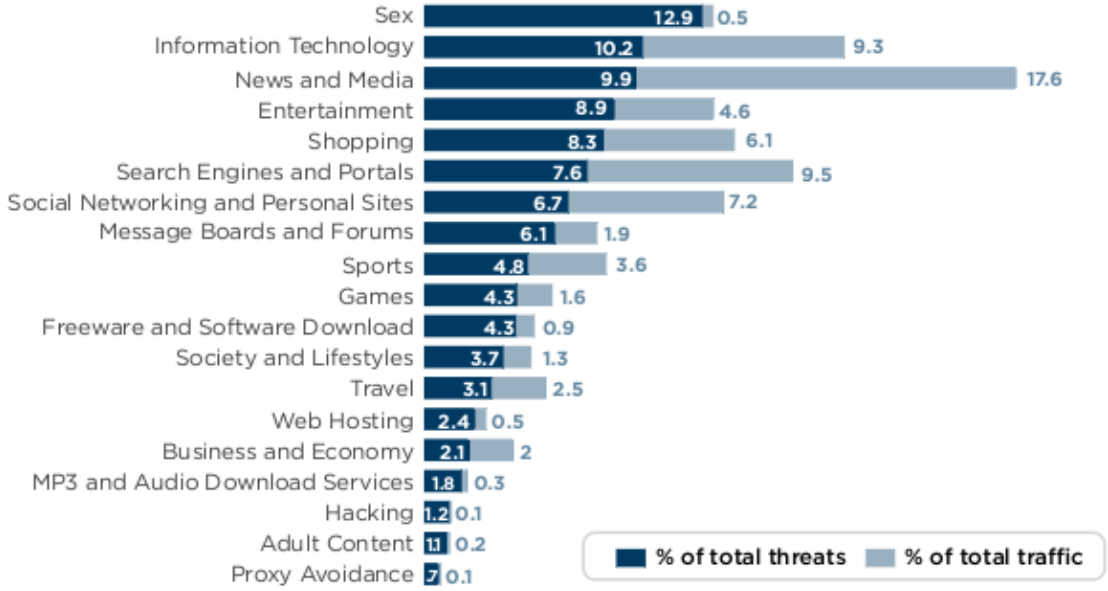

Ilustración 1: Páginas con links maliciosos. [Web Sense (2011), "2010 Threat Report"].

En la *ilustración 2* observamos los porcentajes del contenido al que hacen referencia los mensajes electrónicos considerados como Spam. Como ya se comentó anteriormente en la tabla 2.2 encontramos a la cabeza de estas estadísticas los correos relacionados con las compras, en la que los ciberdelincuentes hacen ofertan suculentas que muchos incautos creen verdaderas.

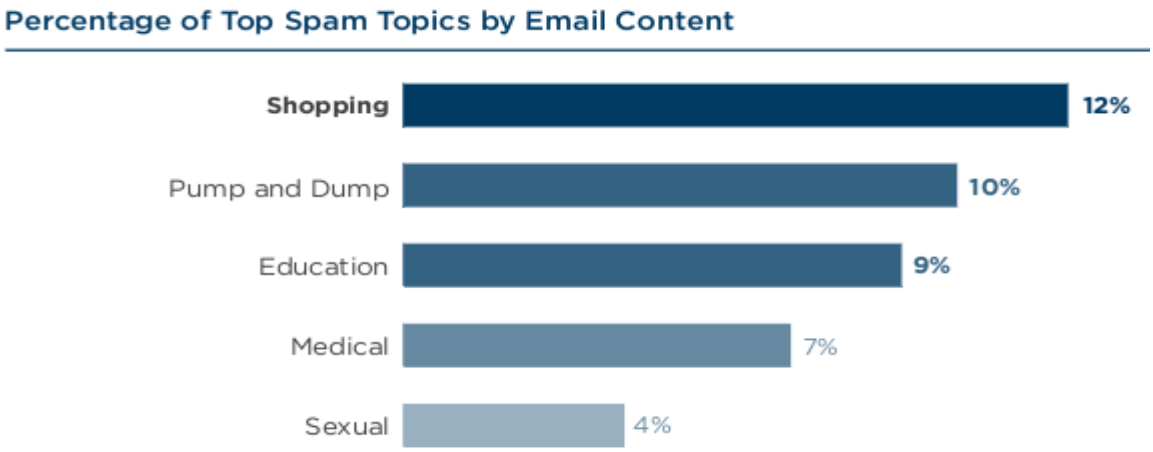

Ilustración 2: Contenido del Spam. [Web Sense (2011), "2010 Threat Report"].

Para finalizar este análisis, vamos a ver algunos datos en cuanto a las costumbres de navegación en las Redes Sociales en el año 2010 (para esto se analizaron 200.000 URLs en Twitter y Facebook), que sin duda los ciberdelincuentes tienen en cuenta a la hora de propagar el malware. En la *ilustración 3* vemos los datos de navegación en Twitter. De lejos la categoría con más movimiento es la relacionada con el *entretenimiento*, seguida con un porcentaje también significativo de *negocios y economía.* En la *ilustración 4* vemos los datos de navegación en este caso de Facebook. También con una gran diferencia la categoría que más movimiento genera es la de *sociedad y estilo de vida,* seguida de lejos por las de *entretenimiento* y *tecnologías*. Aunque a priori los datos de navegación predominantes que acabamos de indicar no son malignos, si que son de interés para analizar la evolución del malware y sobre todo su forma de distribución y propagación, ya que sin temor a equivocación se centrará en las categorías que mueven la mayor cantidad de tráfico.

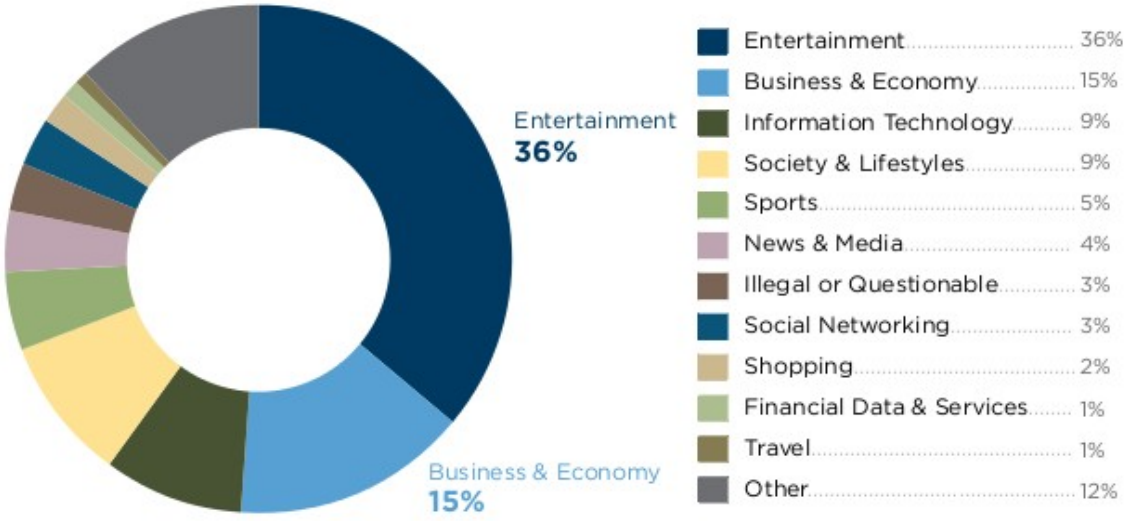

Ilustración 3: Costumbres de navegación en Twitter. [Web Sense (2011), "2010 Threat Report"].

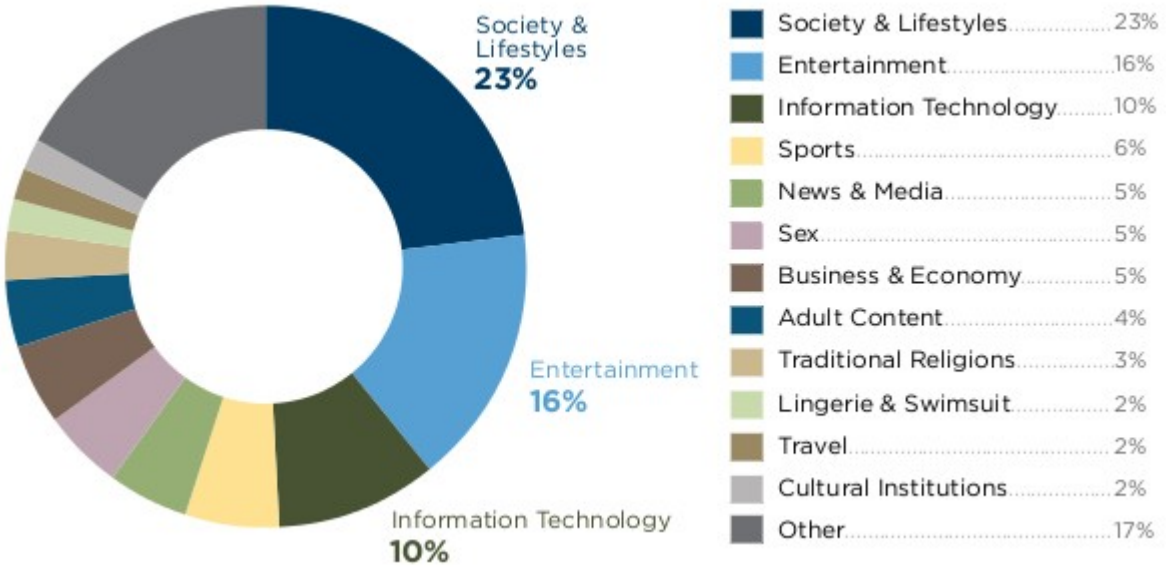

Ilustración 4: Costumbres de navegación en Facebook. [Web Sense (2011), "2010 Threat Report"].

2.2.2.2. Predicciones de amenazas para el 2011

En esta sección vamos a ver lo que se espera para el año 2011, como esperan los expertos que evolucionen las amenazas existentes y se creen nuevas. Para esto vamos a basarnos en el informe de McAfee (McAfee, 2011), categorizando las amenazas que se esperan:

- *Ciberprotestas:* La gran novedad de 2010. La ciberprotesta o ciberactivismo, nuevo movimiento inaugurado por el grupo Anonymous y su Operación Payback, apuntando a objetivos que pretenden acabar con la piratería en Internet primero, y apoyando a Julian Assange, autor de Wikileaks, después, se ha puesto de moda. Incluso usuarios con pocos conocimientos técnicos pueden formar parte de estos ataques de Denegación de Servicio Distribuido (ataques DDoS) o campañas de spam. Aún a pesar de que muchos países están intentado legislar este tipo de actuaciones rápidamente, para poder ser considerada esta actividad un delito y, por lo tanto, perseguida y condenable, se cree que en 2011 veremos proliferar este tipo de cibermanifestaciones.

- *Mac:* Malware para Mac hay, y seguirá habiendo. Crecerá el número a medida que siga aumentando su cuota de mercado. Lo más preocupante es la cantidad de agujeros de seguridad que están apareciendo, ya que los ciberdelincuentes son conscientes de ello y de la facilidad que conlleva estos agujeros de seguridad para distribuir *malware.*

- Dispositivos móviles y Tablets: Se presentan algunos de los cambios más

interesantes del año, en los últimos meses se ha identificado ataques en los nuevos dispositivos tales como los teléfonos inteligentes que utilizan el sistema operativo Android. El *malware* móvil y las amenazas han estado dando vueltas por años, pero ahora tenemos que aceptar que son parte de nuestra vida diaria con los móviles.

- *Redes Sociales:* Las redes sociales han experimentado un auge espectacular en los últimos años. Prueba de ello son los millones de personas que ya han creado su perfil en alguna de ellas o llenado las mismas de contenido variado en lo que se ha decidido llamar la web 2.0. Obviamente, todos estos usuarios no pasan desapercibidos para los creadores de códigos maliciosos quienes, viendo la oportunidad de obtener millones de nuevas víctimas potenciales, no tardaron en empezar a orientar sus creaciones hacia las redes sociales.
  Una de las primeras amenazas que más repercusión tuvo fue el gusano *Koobface*, el cual empezó a propagarse a partir de la segunda mitad de 2008. Este gusano insertaba comentarios que enlazaban a una web donde se almacenaba un supuesto vídeo. Si el usuario accedía a esta web se encontraba con que no podía visualizar el vídeo ya que le hacía falta una supuesta actualización o *codec* de vídeo. El enlace para el supuesto contenía en realidad un enlace para la descarga de *malware*.

- *Amenazas persistentes avanzadas:* Una reciente serie de ataques cibernéticos contra grandes empresas, contratistas del gobierno, instituciones financieras, e incluso proveedores de seguridad, ha puesto de relieve un nuevo tipo de delito: la amenaza avanzada persistente, o APT (*advanced persistent threat*). Esta primavera, estos ambiciosos ataques han afectado a organizaciones tanto con datos valiosos como con los recursos para defenderlos bien, como por ejemplo Google, Citigroup y el Fondo Monetario Internacional. Un reciente ataque de tipo APT efectuado sobre RSA, que ofrece tecnología de seguridad a algunos de los mayores bancos, logró alarmar a los clientes de alto perfil de RSA y parece haber dado lugar a una intrusión en Lockheed Martin, un cliente de RSA.
  A diferencia de las recientes tomas de control de sitios web efectuadas por "*hacktivistas*", o los robos masivos de datos de tarjetas de crédito, las APT son estafas elaboradas y prolongadas difíciles de detectar.

- *Botnets:* Las *botnets* continúan utilizando ordenadores robados y ancho de banda en todo el mundo. Siguiendo el número de caídas con éxito de *botnets*, incluyendo Mariposa, Bredolab y Zeus, los creadores de botnets deben adaptarse a la presión que los profesionales de la ciberseguridad están haciendo sobre ellos. Los laboratorios McAfee Labs predicen que la reciente fusión de Zeus con SpyEye producirá *bots* más sofisticados debido al aumento de mecanismos para evadir la seguridad y aplicación de las leyes. Además, se espera ver una significante actividad de *botnets* en la adopción de funciones de acumulación y extracción de datos, más allá del uso común de enviar *spam*.

2.3. Laboratorios de Seguridad de la Información

Como hemos podido observar en la sección anterior, el número de amenazas y ataques en plataformas electrónicas así como su severidad no deja de aumentar, esto viene en consonancia a la cada vez mayor migración de datos y activos en formato físico a formato electrónico. Teniendo esto en cuenta, se hace imprescindible una concienciación de los usuarios sobre la importancia de tomar una serie de medidas preventivas y buenas prácticas cuando estamos trabajando con tecnologías de la información. Para que esto sea posible debe de haber profesionales lo suficientemente cualificados como para transmitir estos conocimientos al usuario no técnico de una forma amena e intuitiva pero no por ello menos rigurosa. Indudablemente, estos profesionales deben de tener una base técnica que les permita desenvolverse con soltura, esta base en la mayoría de las ocasiones deberá obtenerse en los estudios que ejerzan en su periodo educacional, y más concretamente en las asignaturas relacionadas con la Seguridad de la Información.
Este requisito no es algo nuevo, sino que ha venido repitiéndose desde hace ya bastantes años, por lo que en un primer apartado nos dedicaremos a revisar los *antecedentes docentes* en lo referente al campo de la Seguridad. A continuación revisaremos los *desafíos* a las que nos enfrentamos a la hora de impartir un laboratorio de calidad. Finalmente, discutiremos las diferentes *alternativas existentes* a la hora de implementar un laboratorio, resaltando las virtudes y los defectos de cada una.

2.3.1. Antecedentes en la docencia

Desde principios de los años 90, distintos expertos han recomendado el fortalecimiento de las materias antes mencionadas en las disciplinas de redes e informática. Se ha señalado que la formación a este respecto era insuficiente y que no se ajustaba adecuadamente a las necesidades reales (T.A. Yang, 2004). Para abordar estas necesidades, existen varias alternativas, aunque la más común es dividir la asignatura en dos bloques (UAH, 2008):

- *Seguridad de la Información:* En el primer bloque se aborda el estudio de los mecanismos y servicios para la protección de la información que fluye en las redes de ordenadores. Se abordan las bases matemáticas de la criptografía simétrica y asimétrica y se estudian los fundamentos de los principales algoritmos criptográficos para terminar con una visión general de los mecanismos de autenticación de mensajes, básicamente las funciones resumen y los códigos de autenticación de mensajes.
- *Seguridad de Sistemas:* En el segundo bloque se aborda la problemática de la seguridad desde el punto de vista de los sistemas finales, servidores y elementos de red. Se estudian los aspectos más importantes a tener en cuenta, los ataques más habituales y algunos elementos que permiten aumentar la seguridad de la red como pueden ser los sistemas de detección de intrusiones o los cortafuegos.

En consonancia a la división de los contenidos, los laboratorios de estas asignaturas también se suelen particionar en dos grandes bloques, que se corresponden con los señalados anteriormente. Dado el carácter eminentemente práctico de los conceptos que se imparten, es sumamente importante que la integración entre la parte teórica y la parte práctica sea lo mayor posible, y más aun, que los contenidos que se imparten en los laboratorios tengan un

parecido mínimamente razonable con los escenarios y casos de uso que los alumnos se van a encontrar en el mundo real una vez acaben su formación.

2.3.2. Desafíos

En esta sección vamos a repasar y enumerar los principales desafíos que surgen a la hora de diseñar e implementar un laboratorio de seguridad que sirva a los alumnos para practicar y afianzar sus conocimientos sobre la materia (JIE, 2010):

- *Necesidad de proteger las redes del entorno donde se imparta:* Cuando trabajamos en temáticas relacionadas con la seguridad, es obvio que el número de incidentes relacionados con esta va a aumentar: Los alumnos tratarán con herramientas que se utilizan para realizar auditorías y test de intrusión, *software* vulnerable sobre el que los alumnos practicarán las técnicas que van aprendiendo, malware, etc.
  Sin duda alguna, los administradores de sistemas son reacios a este tipo de uso de sus redes, por lo que tratarán de minimizar estos incidentes al mínimo, en muchos casos aislando los entornos completamente del resto de las redes.

- *Necesidad de acceso desde Internet:* Sería interesante que los alumnos pudieran practicar no solamente en los horarios que tienen asignados para los laboratorios, sino que desde sus propias casas tuviesen acceso al entorno del laboratorio las 24 horas o en periodos que los profesores estableciesen previamente. Si fuese posible conseguirlo, sin duda aumentaría la productividad y la calidad de la enseñanza.
  Este punto choca con el anterior, por un lado nos encontramos con la necesidad de dotar al alumno de la mayor flexibilidad posible, y por el otro con la necesidad por parte de los administradores de sistemas de restringir al máximo el posible uso indebido del entorno.

- *Dificultad para simular entornos de red similares a los existentes en el mundo real:* Si en las asignaturas de seguridad, en las que se supone debemos de adquirir unos conocimientos que nos sirvan para desenvolvernos con naturalidad una vez salgamos del entorno académico y nos centremos en el profesional, los entornos que se nos presentan a la hora de trabajar no tienen una mínima semejanza a los que nos encontraremos en las empresas, la labor de docencia se verá seriamente mermada. Es obvio que en un entorno profesional los recursos disponibles son inmensamente mayores y heterogéneos que en un entorno académico, pero es de vital importancia intentar solventar en la medida de lo posible este punto, ya que si lo conseguimos, el alumno se verá muy beneficiado, siendo capaz de dar el salto de la Universidad al plano profesional sin notar tanto el cambio.

- *Recursos necesarios para desarrollar las prácticas:* Lo ideal es que los alumnos que asistan a estos laboratorios no se vean en la necesidad de llevar su propio material, que en muchas ocasiones será de un costo elevado. Sería impensable que si se pretende emular un escenario con una docena de equipos el alumno se encargue de comprar y configurar este material, por lo tanto, sería deseable que todos los alumnos dispongan de los recursos necesarios para llevar a cabo con éxito las prácticas que se les propongan sin necesidad de que ellos aporten material extra.

- *Empleo de tecnologías acordes al estado del arte:* Una de las máximas que se usa entre los profesionales de las Tecnologías de la Información es *"Renovarse o morir"*, por suerte o por desgracia esta aseveración es totalmente cierta, la rapidez con la que nuevas tecnologías van surgiendo, las existentes se desarrollan, y otras se quedan obsoletas es enorme, por lo que no tendría sentido enseñarle a los alumnos ciertas tecnologías y técnicas que en un corto plazo van a dejarse de usar o que incluso ya lo han dejado de hacer. Teniendo esto en cuenta, los laboratorios que se impartan deben de poder adaptarse a estas nuevas tecnologías y extenderse para poder ofrecer a los alumnos una formación acorde al estado del arte. En resumidas cuentas, dos características imprescindibles para el diseño de los laboratorios de seguridad son la *escalabilidad* y la *extensibilidad*, cualidades sin las cuales la calidad de la enseñanza se vería resentida en un medio/largo plazo.

- *Sobrecarga debida a la configuración y mantenimiento del laboratorio para distintas prácticas o proyectos:* Si el laboratorio que diseñemos debe de servir como base para impartir varias asignaturas, o incluso dentro de la misma diferentes niveles o proyectos, es obvio que el despliegue, configuración y mantenimiento necesarios puede llegar a consumir una gran cantidad de recursos tanto tecnológicos como humanos. Para que esto no sea un impedimento, debemos de diseñar de una manera concienzuda la forma en la que se despliegan los diferentes proyectos a lo largo del laboratorio, como una vez que se realiza esta fase el personal docente lo configura según las necesidades concretas y finalmente realiza el mantenimiento y modificaciones que crea pertinentes. Un punto importante a tener en cuenta, es que estas tres tareas descritas sean lo más automatizables posibles, para evitar tanto fallos humanos como gasto de tiempo innecesario que podría ser empleado en otras tareas que realmente lo requieran.

- *Necesidad de que el alumno pueda adoptar diferentes roles:* Sería deseable que a lo largo del periodo docente en el que se hace uso del laboratorio, el alumno tenga una visión lo más amplia posible de los diferentes roles existentes en el campo de la Seguridad de la Información, para esto, sería interesante que el laboratorio no se enfoque únicamente desde un punto de vista de trabajo cerrado, sino que el alumno pueda trabajar en distintos ámbitos, e incluso que estos diferentes roles puedan ser empleados simultáneamente Un ejemplo con el que podremos entender mejor la importancia de este punto sería el siguiente: la división de los alumnos en tres grupos diferentes, el *primero* de ellos se encargaría de, en un escenario dado, intentar securizar todo el entorno para así obtener la visión del personal de sistemas y seguridad que se encarga de estas tareas. El *segundo* grupo se centraría en una tarea diametralmente opuesta, como podría ser llevar a cabo un test de penetración, de forma que se pondría en la piel de los profesionales que se encargan de realizar tareas de *hacking* ético contra las empresas que requieren sus servicios. Para finalizar, el *tercer* grupo, se encargaría de las tareas de análisis forense, intentando averiguar que brechas de seguridad han aprovechado los atacantes para hacerse con el control del sistema y realizar el debido informe explicándolo detalladamente. Cada uno de estos tres grupos llevaría a cabo su labor de una manera totalmente independiente. Esto es sólo un ejemplo, pero sirve para captar la importancia de que el laboratorio sea lo más flexible posible, para así poder formar a los alumnos en los distintos campos existentes.

2.3.3. Alternativas existentes

Sea cual sea la alternativa que elijamos para nuestro laboratorio, debería de ser capaz de cubrir en la medida de lo posible todos los desafíos anteriormente planteados. Para abordar la tarea de explicar las diferentes opciones existentes, vamos a dividir esttas en tres grandes grupos que nos permitirán entender su funcionamiento:

- *Laboratorio hardware:* La primera opción que se nos viene a la cabeza, y la más realista es tratar con hardware real para montar el escenario que pretendamos usar a la hora de impartir las clases a los alumnos. Los estudiantes, en este tipo de laboratorio, trabajarán con componentes reales, tanto en equipos finales (ordenadores, servidores, etc.), como en equipos de red, por lo que conseguirán un grado de realidad equiparable a lo que se encontrarán a lo largo de su vida profesional. No obstante, este tipo de aproximación se caracteriza por una serie de desventajas que harán raramente posible su implementación:

    - *Coste económico:* El coste asociado a esta aproximación puede hacerse insostenible, el tener que comprar todo el *hardware* necesario para acometer la tarea de montar un escenario mínimamente realista seguramente exceda el presupuesto asignado a cualquier asignatura de seguridad. Por no hablar de que este *hardware* debería de renovarse cada cierto tiempo si nos atenemos al desafío que comentábamos en la sección anterior de mantener el laboratorio acorde al estado del arte, y que si queremos variar los escenarios añadiendo nuevos componentes deberíamos de hacer nuevas inversiones.

    - *Coste temporal:* Otro factor que deberíamos tener en cuenta si usásemos este tipo de laboratorio sería el coste temporal al que deberían de enfrentarse los docentes a la hora de instalas, configurar y mantener todo este *hardware*. Por no hablar del tiempo necesario para volver a reconfigurar los escenarios o desplegar otros nuevos que vayan siendo necesarios.

    - *Portabilidad:* Finalmente, pero no por ello menos importante, otra característica negativa de esta elección sería la imposibilidad de que los alumnos simulen los laboratorios fueran del horario académico que se les asigne para el uso de dichos laboratorios, por lo que se limitaría mucho el trabajo personal que podrían realizar fuera de dichas instalaciones.

  Debido a todo lo expuesto, este tipo de laboratorio, aun proporcionando la experiencia más cercana a la realidad, suele exceder los recursos económicos y humanos disponibles. Como ejemplo de institución que ha optado por este tipo de laboratorio tenemos al *Georgia Tech's Hands-On Information Security Lab* (J.B.G. Randal T. Abler, 2006). Aunque este laboratorio, basado en productos que les proporcionó CISCO, permitía un mínimo grado de automatización, los propios autores aseveraron que la alta inversión económica necesaria hace inasequible esta aproximación para una gran mayoría de instituciones.

- *Laboratorio basado en virtualización descentralizada:* Teniendo en cuenta los problemas que enunciamos en la aproximación de *laboratorio hardware*, la mayoría del personal docente ve imposible el diseño de su laboratorio de la forma descrita. Si recordamos, uno de los principales problema era el coste requerido para adquirir el *hardware* a implantar en el laboratorio. Para paliar esto, podemos beneficiarnos de la tecnología de virtualización, la cual nos permite simular este *hardware* sin el coste asociado que este conlleva. Con este modelo, simplemente deberíamos de valernos de los elementos básicos existentes en los laboratorios para, sobre ellos, montar y simular los diferentes necesarios que los docentes vayan a utilizar. Con la elección de una arquitectura descentralizada, desplegaremos las diferentes máquinas virtuales por todos los equipos existentes en el laboratorio, pudiendo incluso interactuar con otras máquinas no virtuales si así se considera en su diseño. Asimismo, conseguimos una mayor grado de robustez y estabilidad: dado que el estado de las máquinas virtuales puede ser salvado, el estudiante puede trabajar con mayor libertad sobre el escenario. En el caso de que se lleve a cabo alguna acción destructiva voluntaria o involuntariamente, siempre es posible volver al ultimo estado estable que se conserve.
  Hay dos modelos básicos para implementar un laboratorio de este tipo. El modelo que se emplee dependerá de las capacidades que se deseen en la red resultante. En un primer modelo, las imágenes de las máquinas virtuales pueden almacenarse en algún tipo de sistema de almacenamiento centralizado. Cuando el estudiante desee arrancar un proyecto o práctica específico, descargará a su equipo local la imagen apropiada de dicho sistema y arrancará dicha imagen con la tecnología de virtualización disponible. Una ventaja fundamental de este modelo estriba en que se abstrae el *hardware* real de la máquina gracias a la utilización de máquinas virtuales. Otra ventaja importante es que cada estudiante puede reproducir su propio conjunto de máquinas virtuales en un entorno aislado. Como inconvenientes podemos señalar el tiempo necesario para la descarga de las máquinas virtuales en cada uno de los puestos y el tamaño máximo de las redes que pueden simularse que vendría limitado por la capacidad que tenga el equipo local. Un segundo modelo permite el empleo de redes de mayor tamaño con la contrapartida de una menor flexibilidad. En lugar de distribuir un conjunto de imágenes a cada estación de trabajo, un grupo de mayor tamaño de imágenes se distribuye entre un conjunto de estaciones de trabajo. Por ejemplo, suponiendo que cada estación puede albergar 6 máquinas virtuales, podríamos llegar a simular una red de 30 nodos haciendo uso de 5 estaciones de trabajo. Esta aproximación tiene la ventaja de poder agregar los recursos para permitir la simulación de redes de mayor tamaño y acercar la experiencia de trabajar directamente sobre un laboratorio real. De cara a la configuración de red, si la red virtual tiene o una topología plana, es trivial el despliegue del escenario.
  Podemos encontrar ejemplos de este tipo de aproximación en el ReAssure Lab en en Purdue (ReAssure, 2010) y en TinkerNet (T. Winters, 2006).

- *Laboratorio basado en virtualización centralizada:* Una alternativa a la hora de usar la arquitectura distribuida descrita en la sección anterior, es el uso de una aproximación centralizada a la hora de desplegar el escenario deseado. Para conseguir esto, todas las máquinas virtuales se alojarán en un único punto, un servidor central. Aunque en teoría este servidor centralizado podría llegar a soportar virtualización completa o virtualización únicamente del sistema operativo, si lo que

queremos es simular escenarios relativamente complejos, la propia capacidad de cómputo requerida nos limitará típicamente a la simulación de únicamente el sistema operativo. El estudiante podría acceder al servidor central y crear redes de tamaño moderadamente grande con un impacto mínimo sobre dicho servidor. Esto además permitiría que el estudiante pudiera acceder a los escenarios propuestos de forma remota dotándoles de una mayor libertad. Para aumentar las posibilidades de este tipo de laboratorios, es posible emplear no un único servidor sino un *cluster* en el que varias máquinas físicas aparecen como una única máquina lógica. Las principales ventajas que encontramos es una mayor sencillez en el manejo del laboratorio y de acceso para los estudiantes Sin embargo, se depende de una única entidad central, con los riesgos que esto comporta, y es necesario disponer de conectividad con la misma para poder trabajar.

De entre las 3 alternativas expuestas, la que más se aproxima a la elección final es la solución de *virtualización descentralizada*, aunque ligeramente modificada para adecuarla a los requisitos necesarios y que cumpliese en la medida de lo posible los desafíos antes indicados.

2.4. Virtualización

Para poder acometer los desafíos que se nos presentan, es imprescindible el uso de una tecnología novedosa, a la par que potente, como es la virtualización, la cual ha tenido un auge y desarrollo enorme en los últimos años, en gran parte gracias a los importantes ahorros económicos que brinda a las empresas, y la flexibilidad que ofrece. Mediante el uso de ésta, podremos afrontar las tareas necesarias para cubrir en la medida de lo posible los requisitos que enumeramos en el apartado 2.3.2.

En un primer apartado definiremos la Tecnología, explicando las virtudes y defectos que la caracterizan. A continuación discutiremos los diferentes tipos de virtualización existentes, estructurándolas en 3 grandes grupos. Para finalizar, llevaremos a cabo la elección del tipo a usar, para lo cual haremos un estudio previo de las ventajas y puntos flojos de cada uno de los tipos.

2.4.1. Introduciendo la Tecnología

En informática, virtualización se refiere a la abstracción de los recursos de un computador, llamada *Hypervisor* o *VMM* (Virtual Machine Monitor) que crea una capa de abstracción entre el *hardware* de la maquina física (host) y el sistema operativo de la maquina virtual (virtual machine guest), siendo un medio para crear una versión virtual de un dispositivo o recurso, como un servidor, un dispositivo de almacenamiento, una red o incluso un sistema operativo, donde se divide el recurso en uno o más entornos de ejecución (Menken, Ivanka. 2010). El uso de la tecnología de Virtualización nos ofrecerá una serie de ventajas, y como

no, algunos puntos negativos, las cuales pasamos a comentar brevemente:

- *Aislamiento:* las máquinas virtuales son totalmente independientes, entre sí y con el *hypervisor*. Por tanto un fallo en una aplicación o en una máquina virtual afectará únicamente a esa máquina virtual. El resto de máquinas virtuales y el *hypervisor* seguirán funcionando normalmente.

- *Seguridad:* cada máquina tiene una serie de roles de usuario (administrador, usuarios de escritorio, usuarios de servicio, etc) independientes. Por tanto, un ataque de seguridad en una máquina virtual sólo afectará a esa máquina.

- *Flexibilidad:* podemos crear las máquinas virtuales con las características de CPU, memoria, disco y red que necesitemos, sin necesidad de adquirir el *hardware* que cumpla con esas características. También podemos tener máquinas virtuales con distintos sistemas operativos, ejecutándose dentro de una misma máquina física.

- *Agilidad*: la creación de una máquina virtual es un proceso muy rápido, básicamente la ejecución de un comando. Por tanto, si necesitamos un nuevo servidor lo podremos tener casi al instante, sin pasar por el proceso de compra, configuración, etc.

- *Portabilidad:* toda la configuración de una máquina virtual reside en uno o varios ficheros. Esto hace que sea muy fácil clonar o transportar la máquina virtual a otro servidor físico, simplemente copiando y moviendo dichos ficheros que encapsulan la máquina virtual.

- *Recuperación rápida en caso de fallo:* si se dispone de una copia de los ficheros de configuración de la máquina virtual, en caso de desastre la recuperación será muy rápida, simplemente arrancar la máquina virtual con los ficheros de configuración guardados. No es necesario reinstalar, recuperar *backups* y otros procedimientos largos que se aplican en las máquinas físicas.

A pesar de estas virtudes, hay algunos puntos negativos que debemos de soportar si decidimos hacer uso de la virtualización. Todas estas ventajas tienen un precio, que consiste fundamentalmente en una pérdida de rendimiento, es decir, una aplicación generalmente correrá más despacio en una máquina virtual que en un servidor físico. La degradación dependerá de la tecnología de virtualización utilizada, de la configuración realizada a nivel *hypervisor* y de la propia aplicación. Por regla general, las aplicaciones que más repercuten la pérdida de rendimiento son las que realizan operaciones frecuentes de entrada/salida.

Otro aspecto a tener en cuenta es que la máquina física deberá contar con suficiente memoria para poder arrancar todas las máquinas virtuales. Si queremos crear, por ejemplo, 20 máquinas virtuales en un servidor físico y que estén funcionando simultáneamente, hay tecnologías que permiten hacerlo con 1 sola CPU física, como puede ser HP Integrity Virtual Machines (HP, 2011). Pero al menos necesitaremos 1 GB de memoria para cada máquina virtual, más la requerida por el *hypervisor,* lo que daría lugar a unos requerimientos de unos

22 GB de memoria. Es decir, necesitaríamos un servidor con 1 CPU y 22 GB de memoria.

Una vez hemos introducido el concepto básico de virtualización y las bondades que su uso nos ofrece, vamos a pasar a enumerar las diferentes implementaciones de esta tecnología, cada cual con sus pros y sus contras, para finalmente explicar la elección hecha en este Trabajo de Fin de Carrera.

2.4.2. Tipos de Virtualización

Básicamente podemos considerar 3 tipos de virtualización (VMware, 2007): emulación, virtualización completa (*Full Virtualization*), paravirtualización (*Paravirtualization*). Pasemos a describir cada uno de estos con algo más de detalle.

- *Emulación:* La emulación se basa en crear máquinas virtuales que emulan el *hardware* de una o varias plataformas *hardware* distintas. Este tipo de virtualización es la más costosa y la menos eficiente, ya que obliga a simular completamente el comportamiento de la plataforma *hardware* a emular e implica también que cada instrucción que se ejecute en estas plataformas sea traducida al *hardware* real.
  Sin embargo la emulación tiene características interesantes, como poder ejecutar un sistema operativo diseñado para una plataforma concreta sobre otra plataforma, sin tener que modificarlo, o en el desarrollo de firmware para dispositivos *hardware*, donde se pueden comenzar estos desarrollos sin tener que esperar a tener disponible el *hardware* real.

  En la *ilustración 5* podemos observar este tipo de virtualización gráficamente.

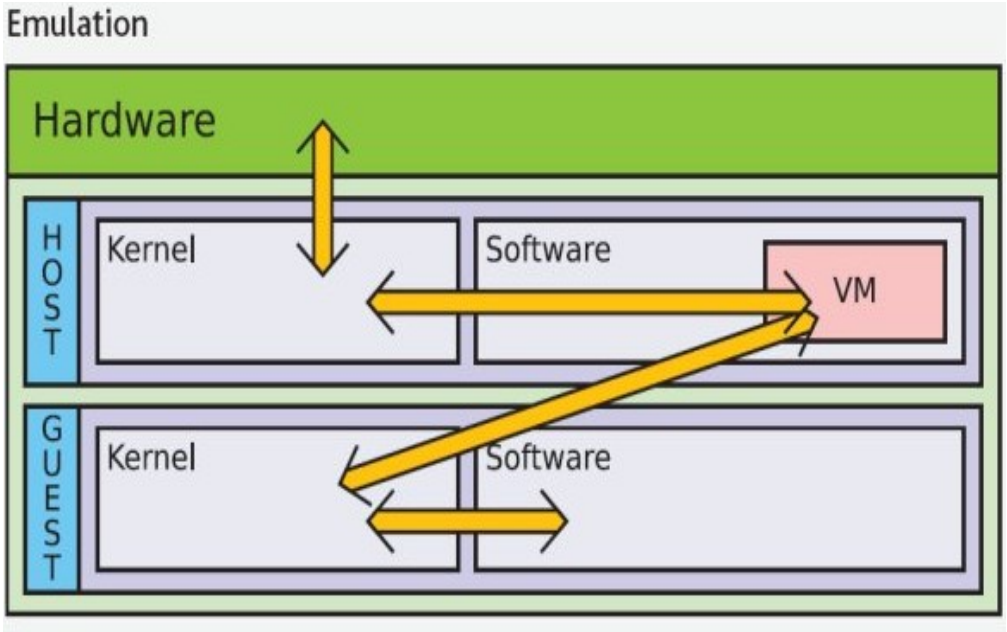

Ilustración 5: Emulación. [Japan Business People technology (2007), "Types of Virtualization"].

Uno de los ejemplos más destacados en la actualidad es QEMU. QEMU, entre otras cosas,

permite emular diferentes plataformas *Hardware* como x86, x86-64, PowerPC, SPARC o MIPS (QEMU, 2011). Así pues, podríamos tener dentro de un servidor linux varios equipos x86 o PowerPC, corriendo diferentes versiones de Linux.

- *Virtualización completa:* Con este término se denominan aquellas soluciones que permiten ejecutar sistemas operativos huésped (*Guest*), sin tener que modificarlos, sobre un sistema anfitrión (*Host*), utilizando en medio un *Hypervisor* o *Virtual Machine Monitor* que permite compartir el *hardware* real. Esta capa intermedia es la encargada de monitorizar los sistemas huésped con el fin de capturar determinadas instrucciones protegidas de acceso al *hardware*, que no pueden realizar de forma nativa al no tener acceso directo a él.

  Su principal ventaja es que los sistemas operativos pueden ejecutarse sin ninguna modificación sobre la plataforma, aunque como inconveniente frente a la emulación, el sistema operativo debe estar soportado en la arquitectura virtualizada.

  En lo que respecta al rendimiento, éste es significativamente mayor que en la emulación, pero menor que en una plataforma nativa, debido a la monitorización y la mediación del *hypervisor*. Sin embargo, recientes incorporaciones técnicas en las plataformas x86 hechas por Intel y AMD, como son Intel VT y AMD-V, han permitido que soluciones basadas en la virtualización completa se acerquen prácticamente al rendimiento nativo (Intel, 2011 y AMD, 2011).

  En la *ilustración 6* podemos observar gráficamente este tipo de virtualización:

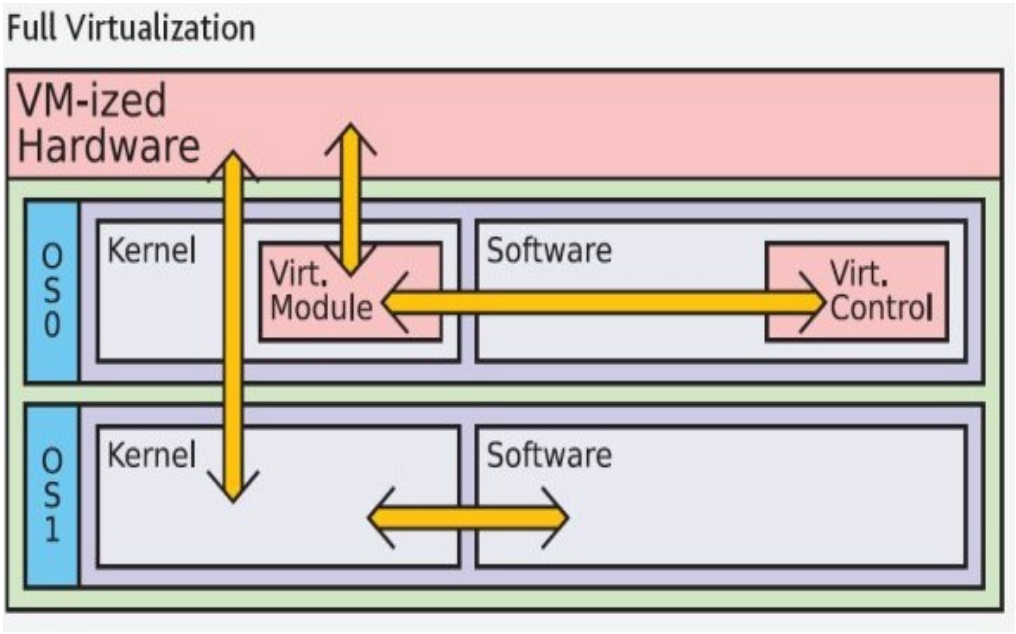

Ilustración 6: Virtualización completa. Emulación. [Japan Business People technology (2007), "Types of Virtualization"].

Un par de ejemplos significativos son VMware[2] y KVM[3].

Hay que tener en cuenta también que la virtualización completa no se refiere a todo el conjunto de *hardware* disponible en un equipo, sino a sus componentes principales, básicamente el procesador y memoria. De esta forma, otros periféricos como tarjetas gráficas, de red o de sonido, no se virtualizan. Las máquinas huésped no disponen de los mismos dispositivos que el anfitrión, sino de otros virtuales genéricos. Por ejemplo, si se dispone de una tarjeta nVidia GeForce en el anfitrión, los equipos huésped no verán esta tarjeta sino una genérica Cirrus.

- *Paravirtualización:* La paravirtualización surgió como una forma de mejorar la eficiencia de las máquinas virtuales y acercarlo al rendimiento nativo. Para ello se basa en que los sistemas virtualizados (huésped) deben estar basados en sistemas operativos especialmente modificados para ejecutarse sobre un *hypervisor*. De esta forma no es necesario que éste monitorice todas las instrucciones, sino que los sistemas operativos huésped y anfitrión colaboran en la tarea.

En la *ilustración 7* podemos ver gráficamente este tipo de virtualización:

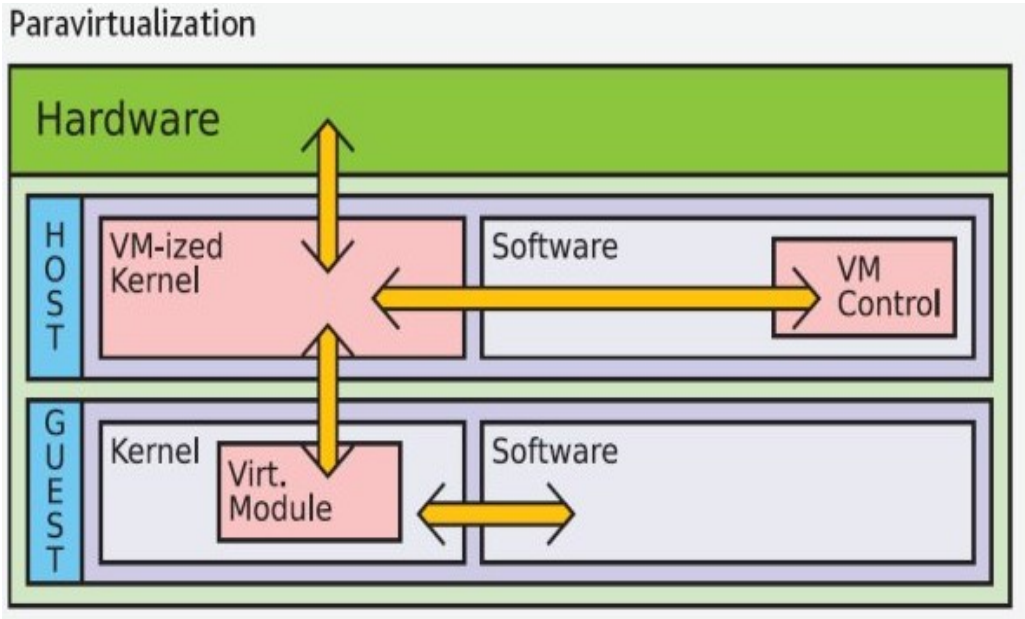

Ilustración 7: Paravirtualización. Emulación. [Japan Business People technology (2007), "Types of Virtualization"].

2 Vmware (2011), http://www.vmware.com/
3 KVM (2011), http://www.linux-kvm.org/page/Main_Page

Uno de los componentes más destacados de esta familia es XEN[4], el cual permite paravirtualización utilizando sistemas operativos modificados, y virtualización completa sobre procesadores con tecnología Intel-VT o AMD-V. Para la gestión de las máquinas virtuales existen aplicaciones propietarias e incluso alguna *open source* como ConVirt[5], que permite gestionar también desde un único sitio las máquinas virtuales de diferentes servidores, realizar tareas sobre ellas, o modificar sus configuraciones.

2.4.3. Elección final

Una vez presentadas las diferentes técnicas de virtualización, pasamos a la elección de una de ellas y a continuación del *software* que nos permita utilizarla. No es una elección fácil, por ello vamos en primer lugar a resumir las principales ventajas de cada aproximación, para después ponerlo en común con los requisitos que son deseables en este Trabajo de Fin de Carrera, de forma que finalmente podremos decidir la técnica de virtualización que más nos convenga.

En la *tabla 4* se pueden ver resumidas las ventajas e inconvenientes de cada una de las técnicas antes descritas.

	Ventajas	**Inconvenientes**
Emulación	- Posibilidad de que los sistemas operativos del host anfitrión y de las máquinas virtuales sean diferentes.	- Costosa y poco eficiente.
Virtualización total	- Los sistemas operativos pueden ejecutarse sin ninguna modificación sobre la plataforma .	- El SO debe estar soportado en la arquitectura virtualizada.
Paravirtualización	- Los sistemas operativos huésped y anfitrión colaboran en la tarea de monitorizar las instrucciones, puede ser muy eficiente.	- Los sistemas operativos guests deben ser modificados para funcionar en este esquema.

Tabla 4: Ventajas e inconvenientes de las 3 técnicas de virtualización

A partir de esta tabla podemos ir sacando conclusiones. La primera de ellas es que el uso de *emulación* no ofrece ventajas significativas con respecto a las otras 2 aproximaciones, y sin embargo se ve seriamente penalizada en cuanto a su rendimiento, por lo que la descartamos. Entre las 2 opciones restantes, y según las ventajas inherentes a cada una, deberíamos de encontrarnos con que el rendimiento de la *paravirtualización* es mejor que con *virtualización total*, pero si nos atenemos a pruebas de rendimiento entre dos soluciones que representen cada una de las técnicas, como puede ser Xen por el lado de la

4 XEN (2011), http://www.xensource.com/
5 ConVirt (2011), http://xenman.sourceforge.net/

paravirtualización y KVM por el lado de la *virtualización total*, vemos que la diferencia no es muy significativa (Xen, 2008) como se observa en la *ilustración 8*:

	Linux	Xen	KVM
CPU	1.000	0.999	0.993
Kernel Compile	1.000	0.487	0.384
IOzone Write	1.000	0.855	0.934
IOzone Read	1.000	0.852	0.994

Ilustración 8: Comparativa de performance entre Xen y KVM. [Xen (2008), "Quantitative Comparison of Xen and KVM "].

Por lo que teniendo en cuenta esto, y que al usar la aproximación de *Virtualización completa* no será necesario hacer ninguna modificación en los Sistemas Operativos tanto anfitriones como huéspedes, esta será nuestra elección final:

- *Tipo de Virtualización:* Virtualización Completa.
- *Solución de Virtualización:* KVM.

2.5. Software libre

Finalmente, vamos a hablar sobre el enfoque elegido a la hora de elegir entre la gran variedad de herramientas disponibles para cada una de las tareas a lo largo de todo el proyecto. Básicamente tenemos 2 grupos en los que dividir el diverso *software* existente:

- *Software* Privativo
- *Software* Libre

Como es fácil discernir por sus nombre, el primero de ellos requiere de una inversión económica precia para poder disfrutar de sus funcionalidades, a veces nos podemos encontrar con versiones demo o trial, las cuales nos proveen de ciertas funcionalidades durante un periodo de prueba, después del cual si queremos seguir usándolo deberemos de abonar su precio. Por contra, el s*oftware* libre no requiere de dicha inversión, puesto que su uso, copia, modificación y distribución es totalmente gratuita. Aunque detrás del *software* libre hay un enorme y complejo mundo de cuestiones morales y/o filosóficas que luego comentaremos brevemente, no se puede negar que el no-coste de éste lo hace muy atractivo. A parte de este punto, hay otro factor que ha hecho que el *software* libre sea el elegido para desarrollar el Trabajo Fin de Carrera en todas sus fases, y ésta es la comodidad del autor con este tipo de *software*, ya que llevo trabajando casi exclusivamente con él desde 8 años aproximadamente, tanto a nivel de aplicaciones de escritorio finales, como a nivel de sistema operativo. Si unimos estos 2 factores a un tercero, que es la realidad de que existen alternativas en el *software* libre para casi la totalidad de aplicaciones privativas, y que

además estas alternativas son de la misma o mayor calidad que las encontradas en el *software* privativo, tenemos las 3 principales razones para justificar la decisión tomada.

A continuación vamos a comentar brevemente alguna de las características ético/morales que identifican este tipo de software, para en el siguiente apartado describir el sistema operativo y las aplicaciones usadas a lo largo del proyecto.

Software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el *software*. De modo más preciso, se refiere a cuatro libertades de los usuarios del *software* (Free Software Foundation, 2009):

- La libertad de usar el programa, con cualquier propósito **(libertad 0)**.
- La libertad de estudiar cómo funciona el programa, y adaptarlo a tus necesidades **(libertad 1)**.
- La libertad de distribuir copias, con lo que puedes ayudar a los demás **(libertad 2)**.
- La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie. **(libertad 3)**.

Un programa es *software* libre si los usuarios tienen todas estas libertades. Así pues, deberías tener la libertad de distribuir copias, sea con o sin modificaciones, sea gratis o cobrando una cantidad por la distribución, a cualquiera y a cualquier lugar. El ser libre de hacer esto significa (entre otras cosas) que no tienes que pedir o pagar permisos.
También deberías tener la libertad de hacer modificaciones y utilizarlas de manera privada en tu trabajo u ocio, sin ni siquiera tener que anunciar que dichas modificaciones existen. Si publicas tus cambios, no tienes por qué avisar a nadie en particular, ni de ninguna manera en particular. La libertad para usar un programa significa la libertad para cualquier persona u organización de usarlo en cualquier tipo de sistema informático, para cualquier clase de trabajo, y sin tener obligación de comunicárselo al desarrollador o a alguna otra entidad específica.

La libertad de distribuir copias debe incluir tanto las formas binarias o ejecutables del programa como su código fuente, sean versiones modificadas o sin modificar (distribuir programas de modo ejecutable es necesario para que los sistemas operativos libres sean fáciles de instalar). Está bien si no hay manera de producir un binario o ejecutable de un programa concreto (ya que algunos lenguajes no tienen esta capacidad), pero debes tener la libertad de distribuir estos formatos si encontraras o desarrollaras la manera de crearlos.
Para que las libertades de hacer modificaciones y de publicar versiones mejoradas tengan sentido, debes tener acceso al código fuente del programa. Por lo tanto, la posibilidad de acceder al código fuente es una condición necesaria para el *software* libre.

Para que estas libertades sean reales, deben ser irrevocables mientras no hagas nada incorrecto; si el desarrollador del *software* tiene el poder de revocar la licencia aunque no le hayas dado motivos, el *software* no es libre. Son aceptables, sin embargo, ciertos tipos de reglas sobre la manera de distribuir *software* libre, mientras no entren en conflicto con las

libertades centrales. Por ejemplo, *copyleft* es la regla que implica que, cuando se redistribuya el programa, no se pueden agregar restricciones para denegar a otras personas las libertades centrales (GNU, 2009). Esta regla no entra en conflicto con las libertades centrales, sino que más bien las protege. *Software* libre no significa no comercial. Un programa libre debe estar disponible para uso comercial, desarrollo comercial y distribución comercial. El desarrollo comercial del *software* libre ha dejado de ser inusual, el *software* comercial libre es muy importante. Pero el *software* libre sin *copyleft* también existe.

3

ANÁLISIS DEL SISTEMA

Una vez que hemos introducido ciertos conceptos básicos, y habiendo explicado el estado actual de las diferentes tecnologías que se van a utilizar a lo largo de este Trabajo de Fin de Carrera, vamos a entrar a analizar los requisitos necesarios para que la aplicación final cumpla con las expectativas puestas en ella.

En la ingeniería de sistemas y la ingeniería de *software*, la ingeniería de requisitos comprende todas las tareas relacionadas con la determinación de las necesidades o de las condiciones a satisfacer para un *software* nuevo o modificado, tomando en cuenta los diversos requisitos de los inversores, en caso de haberlos, que pueden entrar en conflicto entre ellos.

El propósito de la ingeniería de requisitos es hacer que los mismos alcancen un estado óptimo antes de alcanzar la fase de diseño en el proyecto. Los buenos requisitos deben ser medibles, comprobables, sin ambigüedades o contradicciones, etc. (Wikipedia, 2011 a)

En el análisis de requisitos vamos a diferenciar 2 grupos, los cuales serán analizados en sus respectivos apartados:

- Requisitos No Funcionales
- Requisitos Funcionales

En los *requisitos no funcionales* discutiremos las diferentes necesidades *hardware* y *software* del proyecto. Para los requisitos *hardware* debemos adaptarnos en la medida de lo posible al equipamiento ya disponible en un aula de laboratorio. A la hora de analizar los *requisitos funcionales*, en primer lugar estudiaremos los diferentes *casos de uso* que nos podemos encontrar a lo largo del uso de la herramienta y los *diagramas de actividad* que detallan el funcionamiento de estos, para posteriormente poder basarnos en estos a la hora de definir el conjunto de requisitos funcionales. Una vez hecho esto, pasaremos a discutir en diferentes apartados: los diferentes *roles de usuario* con que nos podemos encontrar. También nos encargaremos de analizar el conjunto de *requisitos lógicos* que componen el proyecto (escenarios posibles, ataques, eventos, etc.). Finalmente, debemos dedicarle un apartado al estudio de los *requisitos de seguridad*, ya que debemos asegurarnos que las prácticas del laboratorio no afecten negativamente - de manera directa o indirecta - a componentes externos a éste, así como que los alumnos no puedan modificar o acceder a información que no debería estar disponible para ellos.

3.1. Requisitos No Funcionales

En este apartado vamos a analizar los requisitos no funcionales, esto es, requisitos que ni describen información a guardar, ni funciones a realizar. En concreto vamos a centrarnos en los requisitos *hardware* y a continuación en los requisitos *software*.

3.1.1. Requisitos Hardware

A la hora de definir los diferentes componentes *hardware* que necesitaremos para implementar el proyecto, debemos de tener en cuenta 2 factores:

- Debemos de minimizar la inversión económica en la medida de lo posible, por lo cual tenemos que adaptarnos al material existente en el laboratorio y tratar de aprovecharlo al máximo.
- Debemos ser capaces de hacer frente a los desafíos descritos en el apartado 2.3.2, para poder diseñar un laboratorio totalmente funcional y con el que poder aleccionar a los alumnos de una manera adecuada.

Para conseguir abordar la tarea sin perder de vista estos 2 factores, tenemos que intentar conseguir un equilibrio entre la inversión realizada en los diferentes elementos *hardware* necesarios, y que estos componentes tengan las características mínimas que nos sirvan para enfrentarnos a la labor de dotar al laboratorio de la funcionalidad esperada.

Teniendo esto en cuenta, vamos a pasar a enumerar en la *tabla 5* los elementos necesarios para desplegar el Proyecto de Fin de Carrera.

Nombre	Propósito	Elementos	Cantidad
Ordenador Personal	Equipos para desplegar los escenarios con ayuda de máquinas virtuales.	- Adaptadores de red (NIC, *Network Interface Card*) para conectarse entre sí, idealmente con soporte para un ancho de banda alto (100 Mbps). - El disco duro no es especialmente importante ya que no se prevé un uso excesivo de este. - La RAM y el Procesador deben soportar altas cargas en el movimiento y manipulación de datos. - El procesador debe soportar virtualización, con extensiones como las ofrecidas por Intel VT y AMD-V.	Mínimo 3
Ordenador Personal	Equipo para definir, configurar y mantener los escenarios.	- Adaptador de red (NIC, *Network Interface Card*) para conectarlo a los otros PCs, idealmente con soporte para un ancho de banda alto (100 Mbps). - El disco duro, la RAM y el procesador no son excesivamente importantes, ya que no van a llevar a cabo tareas muy pesadas.	1
Router	Conexión de los PCs del laboratorio entre sí, y con el exterior	- Soporte de un ancho de banda alto (100 Mbps) - Conexión a internet, para el acceso a los escenarios desde el exterior, y para mantener los equipos del laboratorio actualizados.	1

Tabla 5: Requisitos Hardware.

3.1.2. Requisitos Software

Como ya se estableció previamente, la elección hecha consiste en el uso del *software* libre como una de las bases de este Proyecto de Fin de Carrera. El abanico a cubrir es amplio, necesitaremos desde un sistema operativo, pasando por herramientas de control remoto, hasta *software* que nos permita realizar la virtualización. A continuación vamos a analizar todos los elementos *software* necesarios.

Nombre	Sistema Operativo
ID	RNF 1
Descripción	*Software* encargado de ejercer el control y coordinar el uso del hardware entre diferentes programas de aplicación y los diferentes usuarios. Será la base del resto del *software* utilizado posteriormente en los distintos dispositivos *hardware* existentes en el laboratorio.

Nombre	Plataforma de Virtualización
ID	RNF 2
Descripción	Capa de *software* la cual maneja, gestiona y arbitra los cuatro recursos principales de una computadora (CPU, Memoria, Red, Almacenamiento) y así repartir dinámicamente dichos recursos entre todas las máquinas virtuales definidas en la máquina anfitriona.

Nombre	API de Virtualización
ID	RNF 3
Descripción	API la cual brinda una capa de abstracción que nos permite crear administrar y migrar maquinas virtuales independientemente de su virtualizador.

Nombre	GUI de gestión de Virtualización
ID	RNF 4
Descripción	GUI para facilitar la tarea de gestionar las diferentes máquinas virtuales que existirán en el laboratorio.

Nombre	Herramienta de Control Remoto
ID	RNF 5
Descripción	Software que nos brinda la funcionalidad de gestionar máquinas remotas desde una única localización.

Nombre	Suite Retoque Fotográfico
ID	RNF 7
Descripción	Herramienta para la edición de imágenes digitales, ya sean dibujos o fotografías.

Nombre	Captura de Pantalla
ID	RNF 8
Descripción	Aplicación para capturar y editar fácilmente capturas de pantalla.

Nombre	Editor de código fuente
ID	RNF 9
Descripción	Software para la edición del código fuente que compone el proyecto.

A continuación vamos a enumerar en la *tabla 6* la elección hecha para cada uno de los requisitos citados anteriormente, describiendo brevemente las características de esta.

Nombre	Descripción
Sistema Operativo	Linux, distribución **Ubuntu 10.04**[6]. Ha sido el Sistema Operativo elegido como base de todo el trabajo realizado, tanto en mi ordenador personal, como en las pruebas hechas en el laboratorio. Ubuntu es una distribución GNU/Linux que ofrece un sistema operativo predominantemente enfocado a ordenadores de escritorio aunque también proporciona soporte para servidores. Basada en Debian GNU/Linux, Ubuntu concentra su objetivo en la facilidad de uso, la libertad de uso, los lanzamientos regulares (cada 6 meses) y la facilidad en la instalación. Al igual que otros sistemas operativos GNU/Linux, está compuesto de múltiple *software* normalmente distribuido bajo una licencia libre o de código abierto fácilmente obtenible mediante los diferentes repositorios de *software* que se proveen.
Plataforma de Virtualización	**KVM**[7], escogida porque junto a libvirt ofrecen una herramienta altamente potente y extremadamente configurable que cubre las necesidades de este proyecto. Kernel-based Virtual Machine o KVM, (Máquina virtual basada en el núcleo) es una solución para implementar virtualización completa con Linux sobre *hardware* x86. Está formada por un módulo del núcleo (con el nombre kvm.ko) y herramientas en el espacio de usuario, siendo en su totalidad *software* libre. El componente KVM para el núcleo está incluido en Linux desde la versión 2.6.20. KVM permite ejecutar máquinas virtuales utilizando imágenes de disco que contienen sistemas operativos sin modificar. Cada máquina virtual tiene su propio *hardware* virtualizado: una tarjeta de red, discos duros, tarjeta gráfica, etc.
API de Virtualización	**Libvirt**[8]. Esta biblioteca brinda una capa de abstracción y un protocolo que nos permite crear administrar y migrar maquinas virtuales independientemente de su virtualizador y sistema operativo lo cual permite que la virtualización sea mas fácil y homogénea Complemento perfecto para KVM.
GUI de gestión de Virtualización	**Virt-manager**[9] (Virtual Machine Manager) es una interfaz gráfica para la gestión de KVM (y otras muchas plataformas de virtualización), de manera que permite crear y administrar los huéspedes virtualizados, tanto en local como en remoto.
Herramienta de Control Remoto	**OpenSSH**[10] es una versión libre del protocolo Secure Shell (SSH) que es una familia de herramientas para control remoto o transferencia de archivos entre equipos. Las herramientas utilizadas tradicionalmente para realizar estas funciones, eran telnet o rcp, que son inseguras y transmiten la contraseña de los usuarios en texto plano cuando son usadas. OpenSSH proporciona un demonio y unos clientes para facilitar un control remoto seguro y cifrado, así como operaciones de transferencia de archivos, reemplazando de forma efectiva las herramientas heredadas. El componente servidor de OpenSSH, *sshd*, escucha continuamente a la espera de conexiones de clientes desde cualquiera de las herramientas cliente. Cuando aparece una petición de conexión, *sshd* establece la conexión correcta dependiendo del tipo de herramienta cliente que está conectándose. Por ejemplo, si el equipo

6 Ubuntu (2011), http://www.ubuntu.com/
7 KVM (2011), http://www.linux-kvm.org/
8 Libvirt (2011), http://libvirt.org/
9 Virt-manager (2011), http://virt-manager.et.redhat.com/
10 Openssh (2011), http://www.openssh.com/

	remoto se está conectando con la aplicación cliente ssh, el servidor OpenSSH establecerá una sesión de control remoto tras la autenticación. Si el usuario remoto se conecta al servidor OpenSSH con scp, el demonio del servidor OpenSSH iniciará una copia segura de archivos entre el servidor y el cliente tras la autenticación. OpenSSH puede usar muchos métodos de autenticación, incluyendo contraseñas planas, claves públicas y *tickets* de *Kerberos*.
Suite Ofimática	**Libreoffice**[11]. Se trata de una suite ofimática libre y gratuita, que funciona en distintos tipos de ordenadores y sistemas operativos, como por ejemplo Windows, Mac y GNU/Linux. Dispone de un procesador de texto (*Writer*), un editor de hojas de cálculo (Calc), un creador de presentaciones (*Impress*), un gestor de bases de datos (*Base*), un editor de gráficos vectoriales (*Draw*), y un editor de fórmulas matemáticas (*Math*). LibreOffice se creó como una bifurcación de OpenOffice.org en octubre de 2010, y está creada y mantenida por una comunidad liderada por la fundación The Document Foundation. Está disponible bajo la licencia GNU Lesser General Public License.
Suite Retoque Fotográfico	**GIMP**[12] (GNU Image Manipulation Program) es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito. Forma parte del proyecto GNU y está disponible bajo la Licencia pública general de GNU. Es el programa de manipulación de gráficos disponible en más sistemas operativos (Unix, GNU/Linux, FreeBSD, Solaris, Microsoft Windows y Mac OS X, entre otros).
Captura de Pantalla	**Shutter**[13] es un completo capturador de pantalla para sistemas Linux. Cuenta con muchas herramientas, además de las comunes: edición de capturas antes de guardarlas, efectos 3D, *plugins* o sesiones de capturas. Con este programa, podemos capturar la pantalla entera, la ventana de una aplicación, una página web o solamente el área que delimites con el ratón, además de un sin fin de funcionalidades.
Editor de código fuente	**Vim**[14] (del inglés *Vi IMproved*) es un potente y avanzado editor de texto, versión mejorada del editor de texto vi, presente en todos los sistemas UNIX. Su autor, Bram Moolenaar, presentó la primera versión en 1991, fecha desde la que ha experimentado muchas mejoras. La principal característica tanto de Vim como de Vi consiste en que disponen de diferentes modos entre los que se alterna para realizar ciertas operaciones, lo que los diferencia de la mayoría de editores comunes, que tienen un sólo modo en el que se introducen los órdenes mediante combinaciones de teclas o interfaces gráficas.

Tabla 6: Elección del Software.

11 Libreoffice (2011), http://www.libreoffice.org/
12 GIMP (2011), http://www.gimp.org/
13 Shutter (2011), http://shutter-project.org/
14 Vim (2011), http://www.vim.org/

3.2. Requisitos Funcionales

En este apartado vamos a analizar los requisitos funcionales, esto es, los requisitos que nos van a definir *qué* funcionalidad van a estar disponibles para los usuarios que lo usen (Sommerville, Ian, 2006).

En los *requisitos funcionales* haremos una nueva división, en primer lugar estudiaremos los diferentes *casos de uso* que nos podemos encontrar a lo largo del uso de la herramienta, así como los *diagramas de actividad* que definen el funcionamiento de estos, para así poder basarnos en ambos a la hora de definir el conjunto de requisitos funcionales. Empezaremos en un tercer apartado por discernir los distintos *roles de usuario* que debe soportar la aplicación. En un cuarto apartado, analizaremos la amplia variedad de *requisitos lógicos* que compondrán el aplicativo y que explicaremos en profundidad posteriormente, i.e.: los diferentes escenarios posibles, los ataques disponibles, generación de eventos, etc. Para finalizar, pero no por ello menos importante, debemos de ser capaces de cumplir unos *requisitos de seguridad*, para que las prácticas del laboratorio no afecten negativamente de manera directa o indirecta a componentes externos a éste.

3.2.1. Casos de Uso

Un caso de uso describe un conjunto de secuencias, en las cuales cada una de estas representa la interacción de elementos externos sistema (*actores*) con el propio sistema y sus abstracciones (Stevens, Perdita y Pooley, Rob, 2007). Un caso de uso representa uno o más requisitos funcionales completos del sistema, por lo que vamos a pasar a enumerar los casos de uso principales, para así poder definir en los siguientes apartados los correspondientes requisitos funcionales:

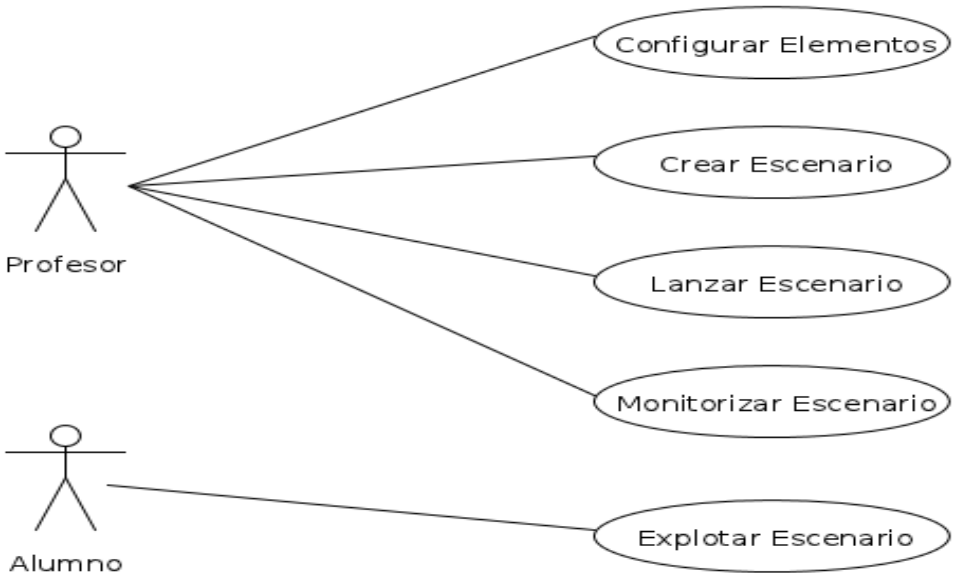

Ilustración 9: Caso de Uso Principal. CU1

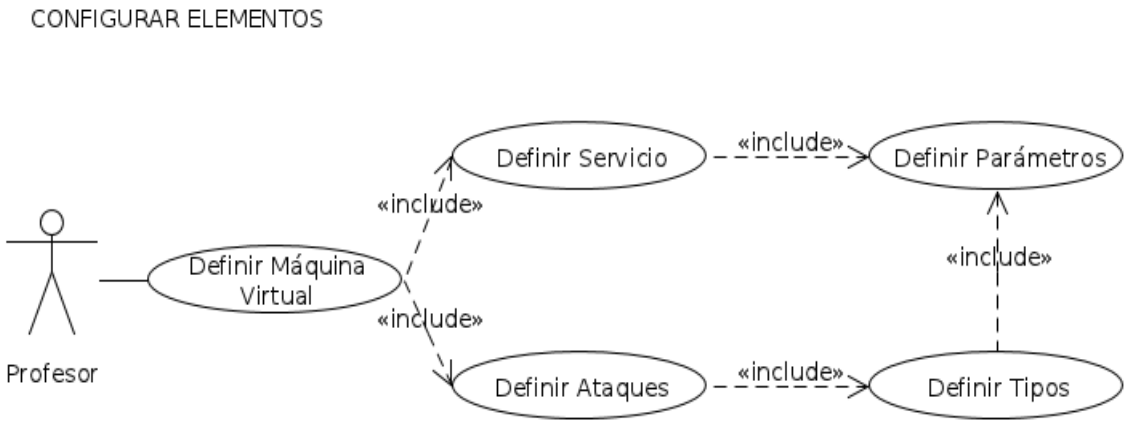

Ilustración 10: Caso de Uso "Configurar Elementos". CU 1.1.

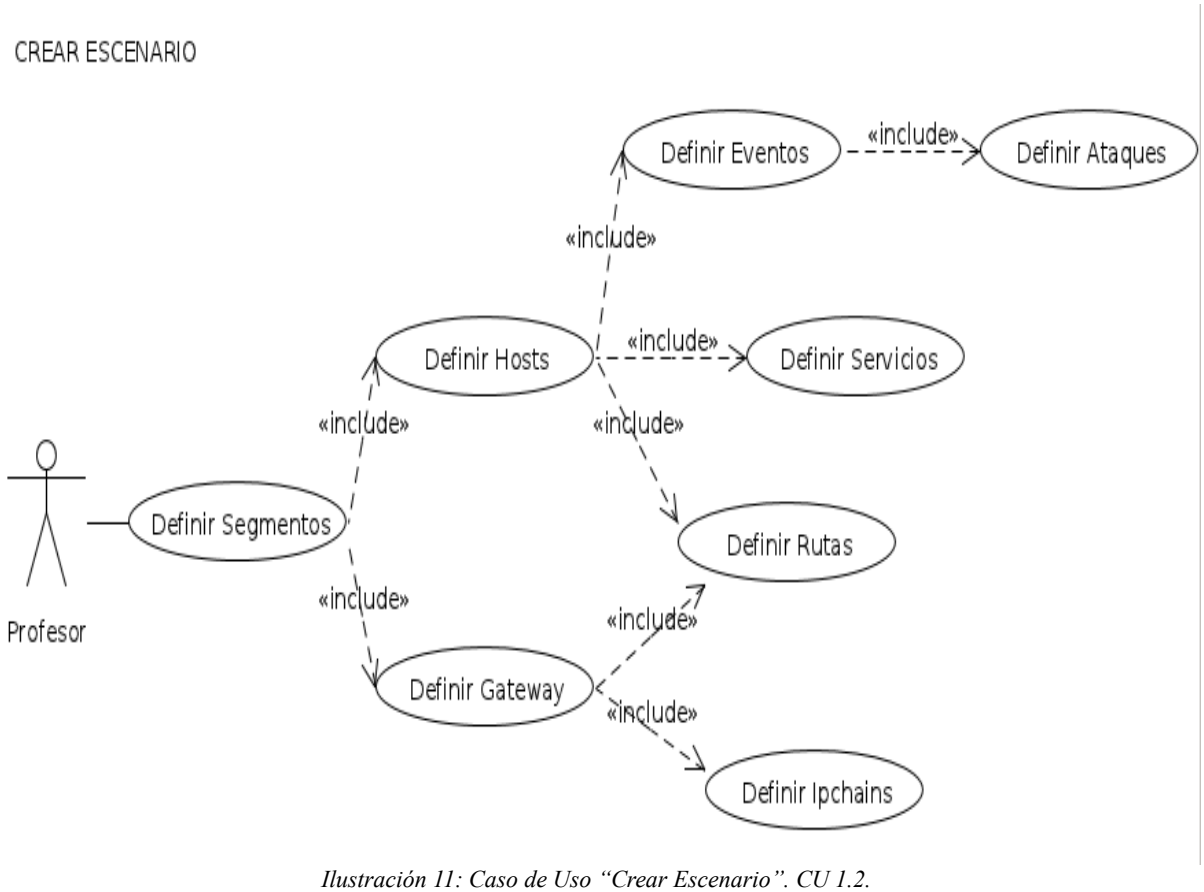

Ilustración 11: Caso de Uso "Crear Escenario". CU 1.2.

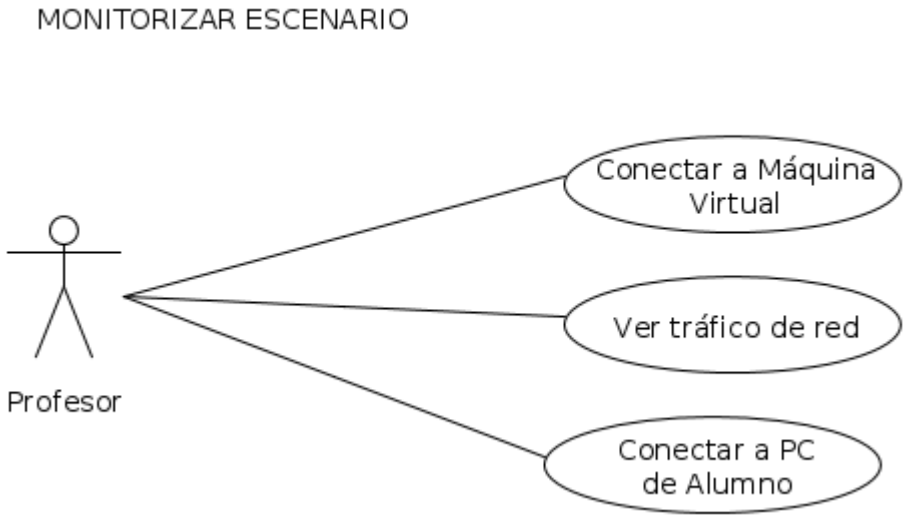

Ilustración 12: Caso de Uso "Monitorizar Escenario". CU 1.3.

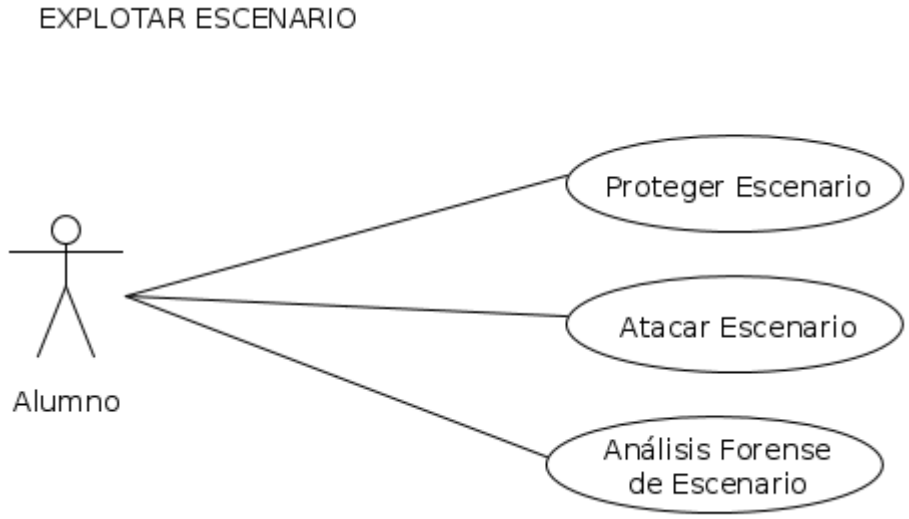

Ilustración 13: Caso de Uso "Explotar Escenario". CU 1.4.

3.2.2. Diagramas de Actividad

Un diagrama de actividad demuestra la serie de actividades que deben ser realizadas en un caso de uso, así como las distintas rutas que pueden irse desencadenando en este (Stevens, Perdita y Pooley, Rob, 2007). Para complementar el apartado anterior, vamos a entrar más en detalle a analizar como debe de actuar la aplicación a la hora de proveer las funcionalidades indicadas en los casos de uso. Realizaremos un diagrama de actividad por cada uno de los casos de uso principales, estos son: *Configurar Elementos, Crear Escenario*, *Lanzar Escenario* y *Monitorizar Escenario*.

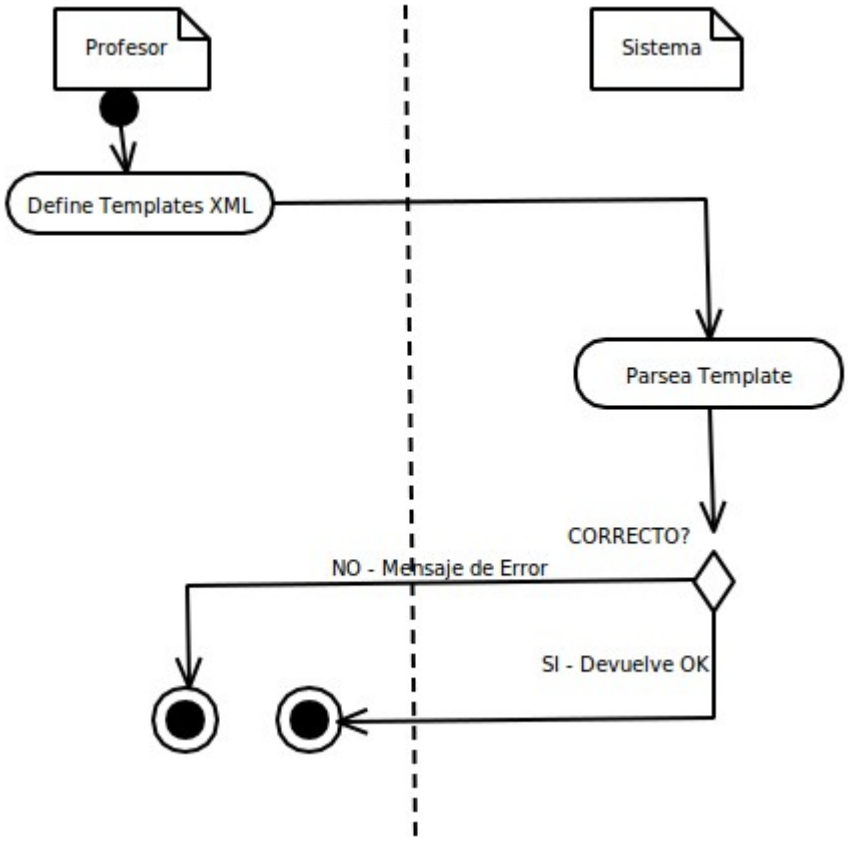

Ilustración 14: Diagrama de Actividad "Configurar Elementos". DA 1.

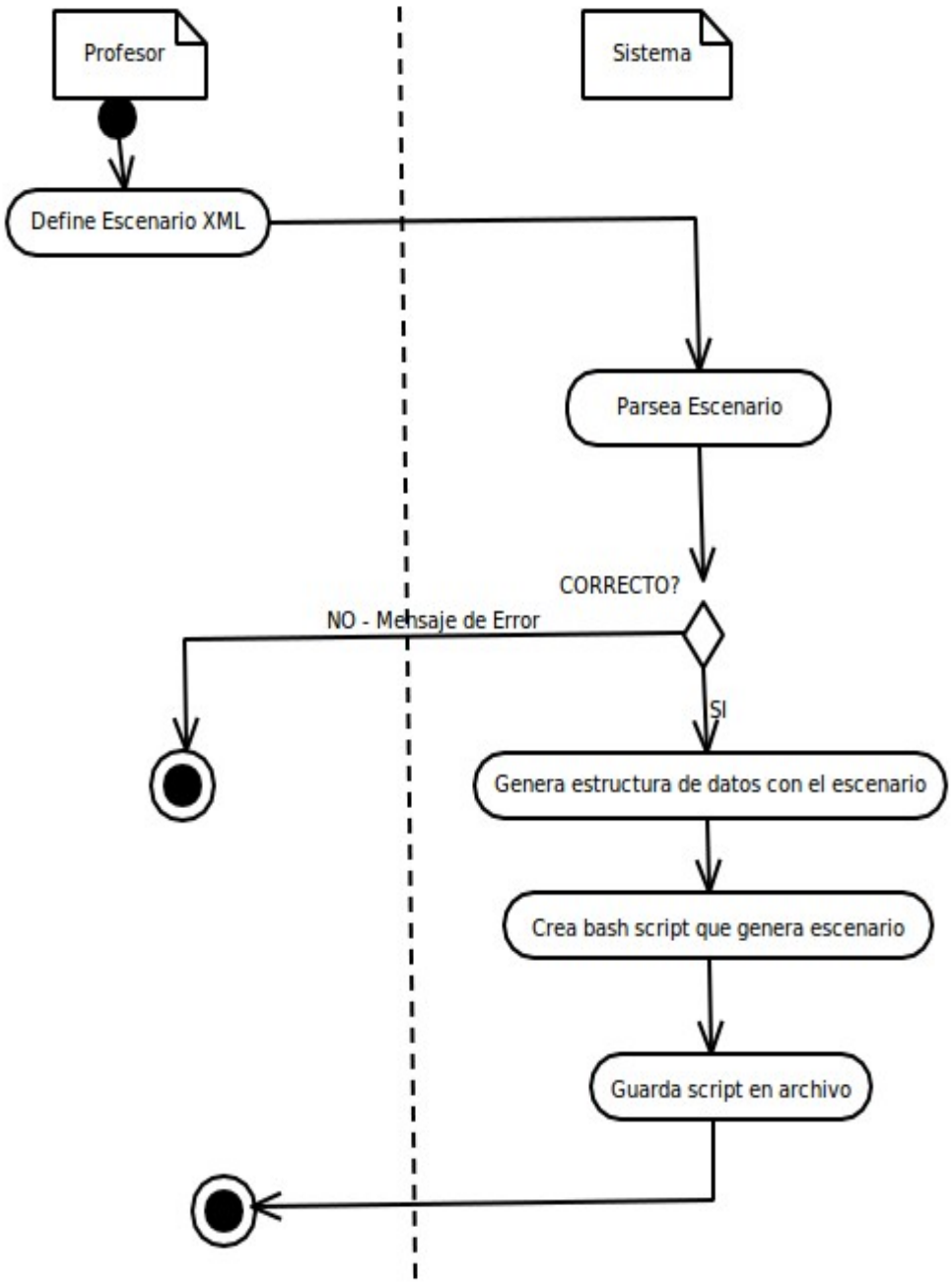

Ilustración 15: Diagrama de Actividad "Crear Escenario". DA 2.

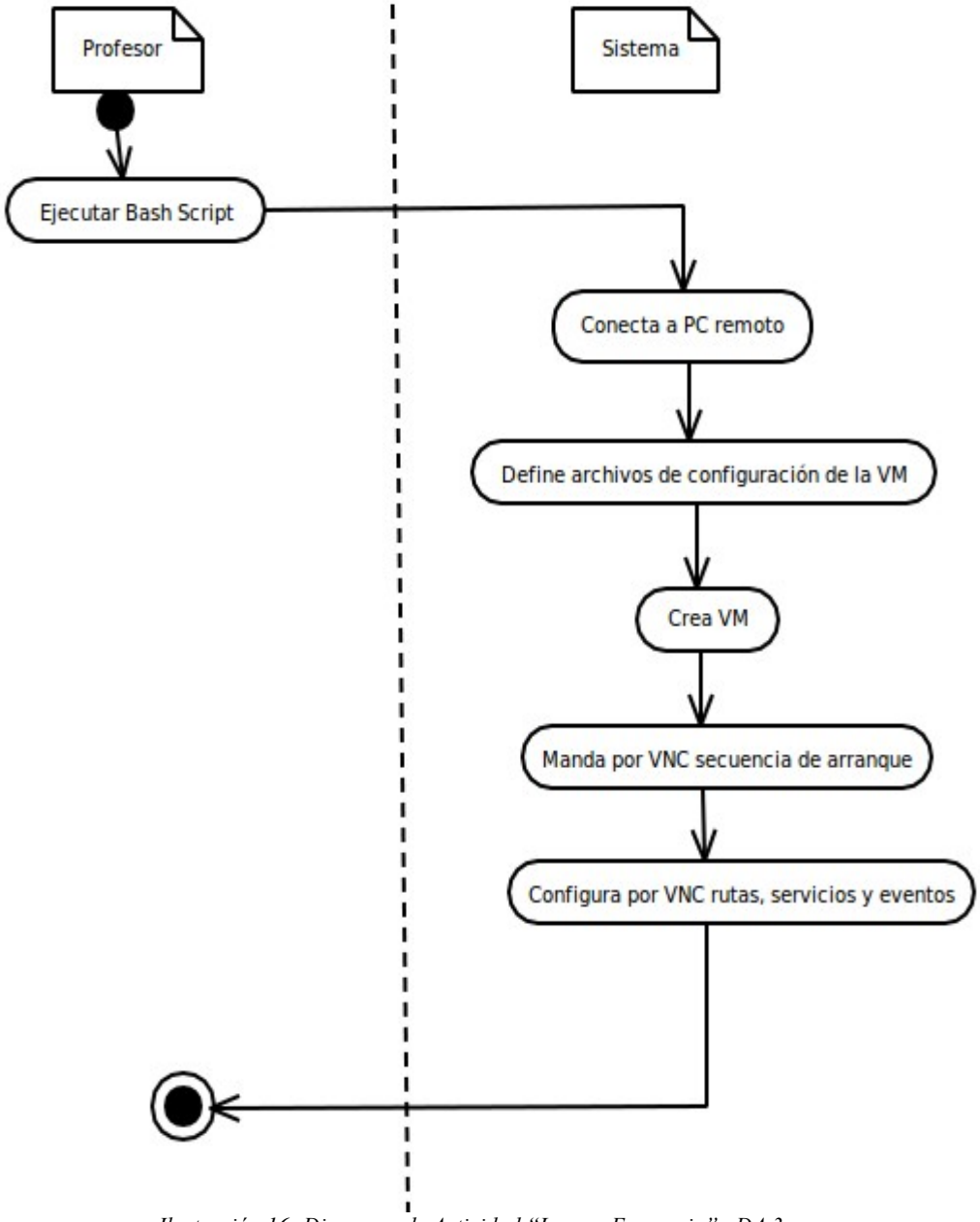

Ilustración 16: Diagrama de Actividad "Lanzar Escenario". DA 3.

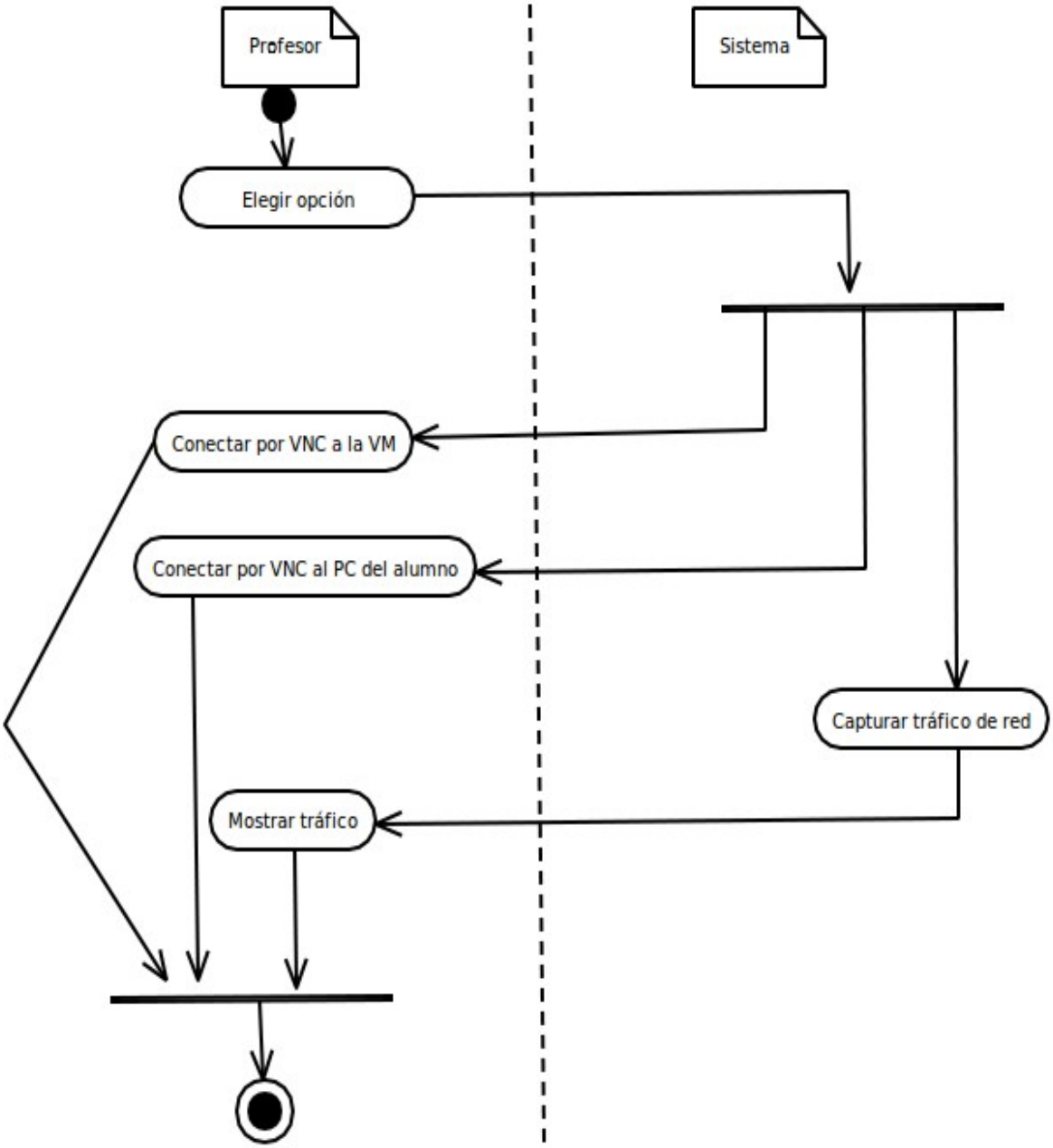

Ilustración 17: Diagrama de Actividad "Monitorizar Escenario". DA 4.

3.2.3. Roles de Usuario

Basándonos en los Casos de Uso y los Diagramas de Actividad de los apartados previos, vamos a pasar a enumerar los diferentes roles de usuario que nos encontraremos en la aplicación:

- *Profesor* (RF1):

Rol encargado de configurar el laboratorio, crear los diferentes escenarios de los que harán uso los alumnos, lanzar estos escenarios para desplegarlos por las diferentes máquinas y finalmente poder monitorizar todo el flujo de información que se cree, tanto entre los propios elementos del escenario, como entre los alumnos y este.

- *Alumno* (RF2):

Rol encargado de consumir los diferentes escenarios creados por los profesores. Los alumnos podrán trabajar con los escenarios de tres formas distintas: atacando, fortificando y finalmente realizando un análisis forense de lo ocurrido.

3.2.4. Requisitos Lógicos

En este apartado vamos a pasar a enumerar los requisitos que componen el núcleo de la aplicación y que darán forma al proyecto. Para ello nos basaremos en los Casos de Uso anteriores.

3.2.4.1. Plantillas

A la hora de crear los escenarios, debemos definir una serie de componentes que lo formarán. Estos componentes estarán en una plantilla configurada por el profesor, de forma que se puedan añadir y modificar fácilmente elementos. Serán los siguientes:

Nombre	Máquina Virtual
ID	RF 3
Descripción	Elemento básico de los escenarios. Serán distribuidas por los diferentes PCs del laboratorio. En las máquinas virtuales debemos ser capaces de definir: *nombre, ruta, plantilla* y *secuencia de arranque*. Dentro de esta podremos incluir: *servicios* y *ataques*.

Nombre	Servicio
ID	RF 4
Descripción	Elemento perteneciente a una máquina virtual. Mediante este describiremos diferentes servicios de sistema accesibles. En los servicios debemos ser capaces de configurar: *nombre* y *comando de inicio*. Dentro de este tendremos: *parámetros*.

Nombre	Ataque
ID	RF 5
Descripción	Elemento perteneciente a una máquina virtual. Mediante este describiremos los posibles ataques disponibles en la máquina virtual. En los ataques debemos de poder configurar: *nombre* y *comando*. Dentro de este tendremos: *parámetros*.

Nombre	Parámetro
ID	RF 6
Descripción	Elemento perteneciente a un servicio o a un ataque. Mediante este describiremos los posibles parámetros de los servicios y de los ataques. En los parámetros debemos de poder configurar: *nombre* y *modificador*.

3.2.4.2. Escenario

Cuando se vaya a configurar un escenario, debemos ser capaces de definir los elementos que lo forman. Estos elementos serán definidos en un archivo creado por el profesor, de forma que será sencillo añadir, modificar y crear nuevos escenarios. Los elementos que deben estar al alcance del profesor son los siguientes:

Nombre	Segmento
ID	RF 7
Descripción	Elemento base del escenario. Agrupará a un conjunto de máquinas virtuales en la misma subred. En los segmentos debemos de poder configurar: *nombre*, *host* anfitrión, *dirección* multicast y *puerto multicast*. Dentro de este tenemos: *host* y *gateway*.

Nombre	Host
ID	RF 8
Descripción	Elemento perteneciente a un segmento. Cada *host* será una instancia de máquina virtual. En los *host* debemos de poder configurar: *ip*, *sistema operativo* y *nombre*. Dentro de este tenemos: *rutas*, *servicios* y *eventos*.

Nombre	Gateway
ID	RF 9
Descripción	Elemento perteneciente a un segmento. Cada *gateway* será una instancia de máquina virtual que conecta 2 segmentos. En los *gateway* debemos de poder configurar: *ip*, *sistema operativo* y *nombre*. Dentro de este tenemos: *rutas* e *ipchains*.

Nombre	Ruta
ID	RF 10
Descripción	Elemento perteneciente a un *host* o a un *gateway*. Cada ruta definirá un camino hacia otra subred. En las rutas debemos de poder configurar: *red* y *gateway*.

Nombre	Servicio
ID	RF 11
Descripción	Elemento perteneciente a un *host*. Mediante este describiremos diferentes servicios de sistema accesibles. En los servicios debemos ser capaces de configurar: *nombre*, *puerto* y *usuario*.

Nombre	Evento
ID	RF 12
Descripción	Elemento perteneciente a un *host*. Mediante este podremos definir eventos a ejecutar en un periodo de tiempo dado. En los eventos debemos ser capaces de configurar: *tiempo*. Dentro de este tenemos: *ataques*.

Nombre	Ipchain
ID	RF 13
Descripción	Elemento perteneciente a un *gateway*. Mediante este describiremos diferentes reglas para el *firewall*. En las *ipchain* debemos ser capaces de configurar: *regla*.

Nombre	Ataque
ID	RF 14
Descripción	Elemento perteneciente a un evento. Mediante este describiremos diferentes servicios de sistema accesibles. En los ataques debemos ser capaces de configurar: *plantilla, tipo* y *parámetros* (deben de ser dinámicos).

3.4.2.3. Configuración

Para poder definir los ficheros descritos en las dos secciones anteriores (plantilla y escenario), es necesario que la aplicación sepa donde encontrarlos, de igual modo que debe saber situar el archivo de salida en el cual escribir el *script* que despliega el escenario y el *script* que manda las instrucciones a través de VNC (como vimos en los diagramas de secuencia). Teniendo esto en cuenta, debemos de facilitar al usuario una forma centralizada de gestionar la configuración, por lo que tendremos:

Nombre	Configuración
ID	RF 15
Descripción	Elemento básico, en el cual podremos definir los siguientes elementos: *ruta de la plantilla*, *ruta del escenario*, *ruta del script de salida*, *ruta del script VNC*.

3.2.5. Requisitos de Seguridad

Como es obvio, no podemos dejar de lado la seguridad. A pesar de ser imprescindible que el laboratorio cubra una serie de bondades descritas en apartados anteriores, no lo es menos el hecho de que al estar trabajando con herramientas de testeo de seguridad, las consecuencias de cara al exterior del laboratorio pueden ser grandes. Debemos en todos los casos paliar cualquier flujo de datos hacia y desde el exterior que no sea estrictamente necesario, y en caso de que lo sea, debe ser dirigido por canales seguros y que no presenten ningún riesgo de cara a la integridad de los sistemas externos al laboratorio.

Además, no tenemos que olvidarnos que el laboratorio servirá para evaluar las habilidades y conocimientos de cada alumno, por lo que debemos asegurar que todos estén en las mismas condiciones, y que ningún alumno se pueda valer del trabajo de otros.

Es por esto, que debemos de ser capaces de cumplir ciertos requisitos de seguridad que enumeramos a continuación:

Nombre	Aislamiento con elementos externos
Descripción	Es necesario que el laboratorio esté aislado del exterior, de modo que sólo entre y salga tráfico por los canales establecidos, esto es, acceso de los alumnos desde instalaciones no pertenecientes a la Universidad a los equipos del laboratorio establecidos para ello.

Nombre	Tráfico del escenario transparente
Descripción	El tráfico transmitido entre los elementos que forman parte del escenario definido por el profesor debe ser totalmente transparente al resto de las máquinas. De no ser así, los alumnos podrían capturar y modificar este flujo de información para su beneficio.

Nombre	Autenticación y comunicación cifrada
Descripción	Cuando los profesores se conecten a las distintas máquinas físicas que alojan las máquinas virtuales, deben hacerlo por medio de canales seguros, tanto para su autenticación, como para el resto del tráfico enviado y recibido.

Nombre	Control de roles de alumno
Descripción	El alumno podrá ejecutar 3 tareas como vimos en los casos de uso. De ninguna forma podrá ejercer dos roles diferentes, ya que la validez de la evaluación podría verse afectada.

4

DISEÑO DEL SISTEMA

Una vez que hemos analizado los diferentes requisitos que componen el proyecto, vamos a centrarnos en hacer un diseño que abarque las necesidades definidas por estos. En la Ingeniería del *Software*, el objetivo del proceso de Diseño del Sistema de Información es la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información (MÉTRICA v.3., 2000). Para abordar esta tarea, vamos a dividir el estudio en varios apartados.

En el primero de ellos, detallaremos la *arquitectura de las máquinas virtuales*, es decir, como va a ser la distribución de éstas a lo largo del entorno del laboratorio, para cubrir las distintas necesidades que se nos plantean. Antes de discutir el diseño final, se introducirán los elementos fundamentales de la arquitectura, los cuales están basados en los requisitos definidos en el capítulo anterior.

En un segundo apartado, presentaremos la sintaxis elegida para la *definición del entorno*. Esto incluye: la definición de las plantillas, de los escenarios y de la configuración - tal y como se explicó en los requisitos del capítulo 3 -. Se explicará detalladamente la sintaxis elegida, dando ejemplos de los distintos componentes.

Otro apartado será dedicado al *despliegue*, en donde se presentará la forma elegida para lanzar los distintos componentes a lo largo del laboratorio. Este apartado estará estrechamente relacionado con el primero, ya que la forma en la que se defina la arquitectura condicionará el *modus operandi* a la hora de desplegar los componentes.

A continuación, explicaremos como será llevada a cabo la *gestión*, otro de los requisitos definidos en el capítulo anterior. Se hablará de las distintas posibilidades que tienen los profesores de interactuar con el escenario una vez esté desplegado.

En el siguiente apartado, hablaremos de la *seguridad*, y de los aspectos relacionados con ésta que se han tenido en cuenta a la hora de realizar el diseño del proyecto. Dada la importancia de este factor, su diseño abarcará varios de los apartados anteriores.

Finalmente, le dedicaremos un apartado al diseño del *software* encargado de usar los diferentes archivos de configuración definidos, para así poder analizar el escenario que se quiere desplegar y ser capaces de generar el *script* que haga esto automáticamente.

4.1. Arquitectura de las máquinas virtuales

Vamos a empezar hablando sobre la arquitectura usada a la hora de tratar con las máquinas virtuales en el entorno del laboratorio. Para poder abordar esta tarea, antes deberemos de fijar los diferentes elementos que formarán parte de los escenarios, y una vez hecho esto, sí que estaremos en disposición de diseñar la arquitectura adecuada.

4.1.1. Elementos fundamentales

Basándonos en los requisitos definidos en el apartado 3.2.4. del capítulo anterior, vamos a pasar a enumerar y explicar brevemente los distintos elementos básicos con los que vamos a trabajar en el proyecto:

- *Host:* Cada una las máquinas finales que compondrán el escenario. Serán altamente configurables y su elección se hará dentro de las opciones existentes en una plantilla, estas dos aseveraciones serán explicadas detalladamente en posteriores apartados.
- *Segmento:* Agrupación de múltiples *host*. Cada uno de estos segmentos contendrá a los diferentes *host* que se encuentren dentro del mismo segmento de red.
- *Gateway:* Tipo particular de *host*. Se usa fundamentalmente con 2 fines: el primero de ellos es unir diferentes segmentos de red, y el segundo es actuar a modo de *firewall*, siendo posible definir reglas para filtrar el tráfico que entra y sale de cada segmento.
- *Ruta:* Regla que define la forma en que un *host* es capaz de enviar tráfico a otro *host* que se encuentren en un segmento de red diferente.
- *Servicio:* En los *host* definidos, podemos lanzar una serie de servicios como pueden ser un *servidor web*, un *servidor ftp*, etc. La elección de estos servicios se hará en función de los servicios definidos dentro de la configuración del *host* en la plantilla correspondiente.
- *Evento:* Se puede definir un conjunto de instrucciones sucesivas a ejecutar por los *host* dentro de un periodo de tiempo determinado.
- *Ataque:* Tipo particular de evento. Existirá un catálogo de ataques definidos en las plantillas, entre los cuales se podrá elegir el deseado.
- *Ipchain:* Regla usada en los *gateway* a la hora de filtrar el tráfico generado entre los diferentes segmentos.

Estos elementos nos sirven como base para ser capaces de definir y explicar la arquitectura de máquinas virtuales en el próximo apartado. Podemos ver una representación gráfica de los diferentes elementos y la forma en que se relacionan en la *ilustración 18*.

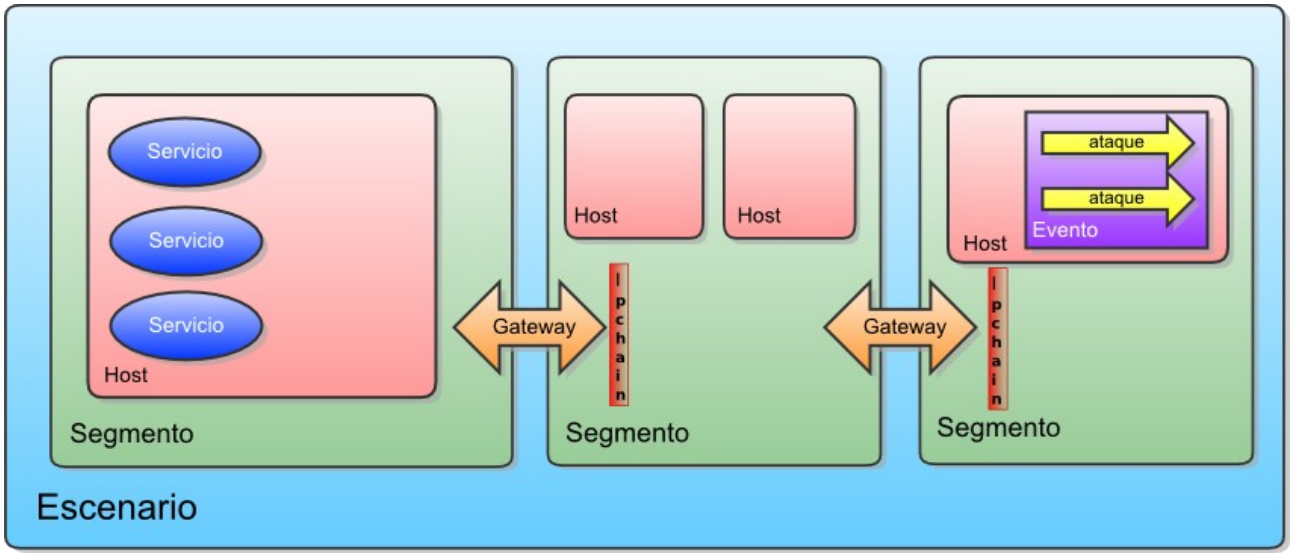

Ilustración 18: Elementos fundamentales. [Enrique de la Hoz,, Iván Marsá-Maestre, y otros., 2010]

4.1.2. Diseño de la arquitectura

Una vez que tenemos presentes los distintos elementos básicos que van a formar parte del laboratorio, estamos en disposición de diseñar la arquitectura a usar. En primer lugar, debemos partir de la base de que el uso de máquinas virtuales para intentar abarcar el *mayor número de escenarios y entornos* diferentes es la mejor opción (J. Smith y R. Nair, 2005), ya que sería impensable tratar de diseñar un laboratorio del tipo aquí expuesto, sobre las máquinas físicas disponibles directamente. A este punto, le unimos que otro requisito indispensable es la *escalabilidad*, es decir, la capacidad del sistema informático de cambiar su tamaño o configuración para adaptarse a las circunstancias cambiantes sin perder calidad en los servicios ofrecidos (André B. Bondi, 2000). Nuestro proyecto debe ser capaz de adaptarse a un entorno cambiante sin que se vea resentida su funcionalidad. Por ejemplo, supongamos que la asignatura ha tenido éxito y el número de alumnos inscritos de un curso lectivo a otro ha crecido en un volumen notable. Para afrontar este crecimiento, los profesores han decidido ampliar el laboratorio comprando unos cuantos equipos más. Nuestra solución tiene que ser capaz de adaptarse a este nuevo escenario, ofreciendo exactamente la misma funcionalidad que antes, sólo que en un entorno con más equipos y alumnos.

Teniendo en cuenta estos detalles, y los ya analizados en el capítulo anterior, la propuesta de la arquitectura final se basa en el uso de máquinas virtuales distribuidas a través de las máquinas físicas existentes en el laboratorio. De este modo, y gracias al uso de la plataforma de virtualización KVM, cada máquina física disponible será la encargada de soportar (*host anfitrión*) uno o más segmentos de red, cada uno de los cuales contendrá una o más máquinas virtuales. Cada segmento de red existente, se conectará al resto de los segmentos mediante el uso de *gateways*, también virtuales. De este modo los *host* no percibirán en

modo alguno el hecho de que el resto de *host* estén en máquinas físicas diferentes. A continuación, en la *ilustración 19*, se muestra un ejemplo gráfico de esto.

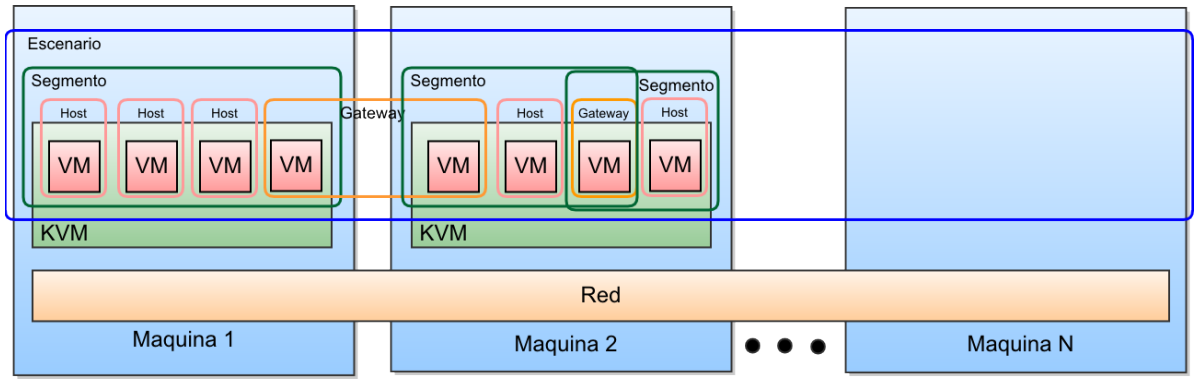

Ilustración 19: Arquitectura distribuida de máquinas virtuales. [JIE, 2010]

Con el uso de una arquitectura virtual como la que se encuentra en la figura, conseguimos abordar los 2 requisitos presentados al principio de este apartado. Por una parte, mediante el uso de máquinas virtuales de una manera distribuida, hacemos que nuestros escenarios puedan ser mucho más completos, flexibles y eficientes que sin su uso. Esto es debido a que se puede repartir la carga computacional total que supone desplegar un escenario, entre los diferentes elementos físicos del laboratorio. Y por otro lado, si fuera necesario ampliar el número de equipos físicos en el laboratorio, simplemente se debería de desplegar en él los elementos del escenario definido que creamos necesario, exactamente como se hace con el resto de los equipos físicos en un primer momento.

En la podemos observar que por debajo de toda la arquitectura diseñada, tenemos la propia red del laboratorio, que a fin de cuentas será la encargada de transmitir el tráfico entre los diferentes elementos. Pero sobre este soporte de red, debemos ser capaces de simular segmentos de red para así configurar correctamente las máquinas virtuales. La solución a esto nos la provee *libvirt* y su *multicast tunnel* a la hora de definir los interfaces de red (libvirt, 2011). Usando este tipo de interfaz de red virtual, definimos grupos *multicast* que representan redes virtuales. Cualquier máquina virtual que tenga sus interfaces de red en el mismo grupo *multicast* puede hablar con las demás a través de la propia infraestructura de red física, incluso estando en máquinas físicas diferentes. Aunque este modo de trabajar nos facilita mucho la implementación del proyecto, posee algunas características que deberemos tener en cuenta a la hora de diseñar el resto del proyecto:

- No hay soporte *DHCP*. Por lo que deberemos gestionar la configuración de las direcciones *IP* en cada máquina virtual que se levante.

- No hay acceso a la red externa. Por lo que si queremos esta funcionalidad, debemos encargarnos de configurar una máquina virtual con un segundo interfaz de red que nos provea el acceso al exterior.

A parte de lo ya explicado, la arquitectura elegida plantea una serie de desafíos, que iremos desgranando en apartados posteriores. Algunos de estos son el control de arranque y parada de las instancias de cada máquina virtual, el aislamiento adecuado de los diferentes segmentos de red, el cifrado del tráfico intercambiado entre los elementos del escenario, etc.

4.2. Definición del entorno

Una vez que hemos mostrado la arquitectura elegida a la hora de implementar el proyecto, vamos a centrarnos en la forma en la que los profesores definirán todo el entorno, esto es: p*lantillas*, *escenarios*, configuración *de archivos*, *plantilla de máquinas virtuales* y *configuración de secuencia de arranque de las máquinas virtuales*. En el capítulo anterior ya se especificó, en la definición de requisitos, los componentes necesarios en cada uno de los elementos, por lo que ahora nos centraremos en mostrar con ejemplos la forma de describirlos.

Se ha optado por el uso de XML (*eXtensible Markup Language,* Lenguaje de Marcas Extensible), el cual es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium*(W3C)[15], que permite definir la gramática de lenguajes específicos, en este caso el usado a la hora de definir las plantillas y los escenarios. Mediante el uso de XML, conseguimos algo muy importante, que la definición de cada uno de los elementos sea *extensible*, de forma que será sumamente sencillo añadir o modificar nuevos elementos.

4.2.1. Plantillas

El primer paso para configurar el proyecto será la definición de una plantilla, en la cual figurarán todas las máquinas posibles que estén disponibles a la hora de definir un escenario.

En la *ilustración 20* podemos observar un ejemplo de archivo *template:*

```
<?xml version="1.0"?>

<!-- Archivo donde definimos las plantillas de los OS
existentes: rutas, servicios, parámetros, etc... -->


<templates>

        <vm name="dvl" path="/home/antonio/Escritorio/PFC/DVL.iso"
        template="/home/antonio/Escritorio/PFC/dvl.xml"
boot="/home/antonio/Escritorio/PFC/dvl.boot">

                <service name="www" init="start-apache" >

                        <param name="port" mod="-p" />

                        <param name="user" mod="-u" />

                </service>

                <attack name="portscan" command="nmap" >

                        <type name="OS" parameter="-O" >

                                <param name="dst" mod=" " />

                        </type>

                </attack>

        </vm>

</templates>
```

Ilustración 20: Ejemplo de Templates.xml

15 W3C (2011), http://www.w3c.es/

En este archivo de ejemplo se ha definido una única *máquina virtual*, llamada *dvl*. La máquina virtual puede correr un *servicio* - *Servidor Web* -, en el que podremos definir el *puerto* donde se pondrá a la escucha y el *usuario* que lo lanzará. También tendremos disponibles un *ataque*, e*scaneo de puertos*, con un único *tipo*, *escaneo del Sistema Operativo*, el cual no requiere ningún parámetro extra.

Vamos a comentar la sintaxis:

- El *tag* principal será *templates*.
- Dentro definiremos cada máquina virtual mediante el *tag vm*. Contendrá los siguientes atributos:
  - *Name:* Nombre de la máquina virtual
  - *Path:* Ruta de la ISO
  - *Template:* Ruta a la definición de la configuración para su uso con KVM
  - *Boot:* Ruta al archivo que contiene la secuencia de arranque
- La máquina virtual puede contener servicios, mediante el *tag service*. Los atributos serán:
  - *Name:* Nombre del servicio
  - *Init:* Comando para arrancar el servicio
- Cada servicio tendrá una serie de modificadores, definidos mediante el *tag param*. Sus atributos serán:
  - *Name:* Nombre del parámetro
  - *Mod:* Modificador para configurar el parámetro
- La máquina virtual puede contener ataques, mediante el *tag attack*. Los atributos serán:
  - *Name:* Nombre del ataque
  - *Command:* Comando para lanzar el ataque
- Cada ataque tendrá distintos tipos, definidos mediante el *tag type*. Sus atributos serán:
  - *Name:* Nombre del tipo de ataque
  - *Parameter:* Parámetro a usar dentro del ataque
- Cada tipo de ataque tendrá una serie dinámica de modificadores, definidos mediante el *tag param*. Los atributos serán:
  - *Name:* Nombre del modificador
  - *Mod:* Modificador para configurar el parámetro

4.2.2. Escenarios

Una vez que hemos definido en el *templates.xml* las diferentes opciones de las máquinas virtuales existentes, podemos definir ya el escenario a desplegar. En la *ilustración 21* observamos un ejemplo de archivo *stage.*

En este escenario de ejemplo se han definido 3 *segmentos* de red diferentes, con 2 *gateways* que los conectan. Dentro de cada segmento de red hay una serie de *host,* con sus respectivas *rutas, servicios* y *eventos* programados. Además de esto, los *gateways* también tienen definidos sus *rutas*, y las *ipchains* para configurarlos a modo de *firewall.*

```
<?xml version="1.0"?>
<!-- Escenario con 3 segmentos de red, y 2 GWs que los interconectan -->
<stage>
<segment label="lan1" host="192.168.1.128" mcast="230.0.0.1" port="8888">
 <host ip="192.168.2.50" template="dvl" name="dvl1">
        <route net="192.168.3.0/24" gw="192.168.2.1" />
        <route net="192.168.4.0/24" gw="192.168.2.1" />

        <service name="sshd" port="80" user="root" />
        <service name="ftp" port="80" user="root" />

        <event at="1minutes">
                <attack template="portscan" type="tcp-syn" dst="192.168.4.50" />
                <attack template="ping" type="icmp" dst="192.168.3.50" />
         </event>
 </host>
</segment>

<segment label="internet" host="192.168.1.129" mcast="230.0.0.1" port="8889">
 <gateway template="firewall" ipin="192.168.2.1" ipout="192.168.3.1" linkto="lan1"
name="gw1" >
         <route net="192.168.4.0-24" gw="192.168.3.2" />
         <ipchains rule="-t filter -A INPUT -p tcp --dport 25 -j ACCEPT " />
         <ipchains rule="-t filter -A INPUT -p tcp --dport 80 -j ACCEPT " />
 </gateway>
 <host ip="192.168.3.50" template="dvl" name="dvl2">

        <route net="192.168.2.0/24" gw="192.168.3.1" />
        <route net="192.168.4.0/24" gw="192.168.3.2" />
        <service name="www" port="80" user="root" />
        <service name="pgsql" port="80" user="root" />

        <event at="5minutes">
                <attack template="portscan" type="OS" dst="192.168.2.50" />
                <attack template="ping" type="icmp" dst="192.168.2.50" />
        </event>
 </host>
</segment>

<segment label="dmz" host="192.168.1.130" mcast="230.0.0.1" port="8890">
 <gateway template="firewall" ipin="192.168.3.2" ipout="192.168.4.1" linkto="internet"
name="gw2" >
         <route net="192.168.2.0/24" gw="192.168.3.1" />

         <ipchains rule="-t filter -A INPUT -p tcp --dport 80 -j ACCEPT " />
         <ipchains rule="-t filter -A INPUT -p tcp --dport 25 -j ACCEPT " />

 </gateway>
 <host ip="192.168.4.50" template="dvl" name="dvl3">

         <route net="192.168.2.0/24" gw="192.168.4.1" />
         <route net="192.168.3.0/24" gw="192.168.4.1" />

         <event at="2minutes">
                <attack template="portscan" type="port" ports="80,443" dst="192.168.2.50"
/>
                <attack template="portscan" type="tcp-connect" dst="192.168.4.50" />
        </event>
 </host>
</segment>

</stage>
```

Ilustración 21: Ejemplo de Stage.xml

Vamos a comentar la sintaxis:

- El *tag* principal es *stage*
- Dentro definirimos una serie de segmentos mediante el *tag segment*. Que tendrá los siguientes atributos:
  - *Label:* Nombre del segmento
  - *Host:* Dirección IP del equipo físico sobre el que desplegarlo
  - *Mcast:* Dirección de *Multicast*
  - *Port:* Puerto de *Multicast* *
- Un segmento puede tener PCs dentro, definidos por el *tag host*. Con los siguientes atributos:
  - *Ip:* Ip que se le asignará
  - *Template:* Nombre de la máquina virtual en el archivo *templates.xml*
  - *Name:* Nombre que se le asignará al equipo
- El equipo puede tener servicios, definidos mediante el *tag service*. Con los siguientes atributos:
  - *Name:* Nombre del servicio en el archivo *templates.xml*
  - *Port:* Puerto en el que lanzar el servicio
  - *User:* Usuario con el que arrancar el servicio
- El equipo puede tener eventos, definidos mediante el *tag event*. Con el siguiente atributo:
  - *At:* Instante de tiempo en el que lanzar el evento**
- El evento contiene ataques, definidos mediante el *tag attack*. Con los siguientes atributos:
  - *Template:* Nombre del ataque en el archivo *templates.xml*
  - *Type:* Tipo del ataque en el archivo templates.xml
  - Parámetros: Conjunto dinámico de parámetros.
- Un segmento puede tener *gateways* dentro, definidos por el *tag gateway*. Con los siguientes atributos:
  - *Template:* Nombre de la máquina virtual en el archivo *templates.xml*
  - *Ipin:* Ip que se le asignará al primer interfaz de red (segmento 1)
  - *Ipout:* Ip que se le asignará al segundo interfaz de red (segmento 2)
  - *Linkto:* Nombre del segmento a conectar***
  - *Name:* Nombre que se le asignará al *firewall*
- Un *firewall* puede tener reglas para filtrar, definidas por el *tag ipchain*. Con el siguiente atributo:
  - *Rule:* Valor de la regla****
- Un *host* y un *firewall* pueden tener rutas, definidas por el *tag route*. Con los siguientes atributos:
  - *Net:* Dirección IP y máscara de la red
  - *Gw:* Dirección IP del *gateway* por el que mandar el tráfico

---

* Para definir un grupo *Multicast* lo hacemos mediante la tupla (Dirección *Mcast*, Puerto *Mcast*), variando cualquiera de estos valores cambiaremos el grupo.

** El formato de *at* usado es el indicado en las páginas *man:* http://linux.die.net/man/1/at
El profesor podrá introducir las siguientes unidades: ***minutes, hours, days, weeks***.

*** En el ejemplo el firewall se ha definido en el segundo de los segmentos que se pretenden conectar, y por lo tanto el valor del atributo *linkto* será el nombre del primer segmento.

**** El formato usado es el indicado en las páginas *man* de *iptables*: http://linux.die.net/man/8/iptables

4.2.3. Configuración de archivos

A la hora de usar la herramienta desarrollada en este proyecto, es necesario definir una serie de parámetros: *ruta del archivo templates.xml*, *ruta del archivo stage.xml*, *ruta del archivo con el script de salida* y *ruta del script que manda instrucciones por VNC*.

Para que no sea necesario cada vez que ejecutamos la herramienta introducir todos estos parámetros, la solución diseñada ha sido el uso de un archivo de configuración -*config.cfg-,* donde definiremos todos estos atributos, y del cual leera nuestra herramienta para coger los valores cuando los necesite.

A continuación observamos en la *ilustración 22* un ejemplo de este archivo:

```
[main]

vnc_sendkeys=./vnc_send_keys.py
templates=./templates.xml
input_stage=./escenario.xml
output_script=./salida.sh
```

Ilustración 22: Ejemplo de Stage.xml

Comentemos la sintaxis*:

- En primer lugar debemos definir una sección llamada *main*
- A continuación cada par atributo-valor

4.2.4. Plantillas de máquinas virtuales

Como pudimos observar a la hora de definir las máquinas virtuales en el archivo *templates.xml*, cada una de éstas tiene un atributo llamado *template*, con este atributo indicamos la ruta del archivo que define la configuración XML** de dicha máquina virtual, para que *libvirt* sea capaz de crear una instancia mediante KVM.

A la hora de definir este archivo tenemos disponibles los distintos elementos que aparecen detalladamente explicados en la documentación de *libvirt, pero* a pesar de la flexibilidad que nos brinda *libvirt*, debemos de tener en cuenta una serie de puntos para que la configuración sea correcta:

- Debemos de poder personalizar dinámicamente una serie de atributos: *name, uuid, file_disk, file_iso, mcast* y *port.* Para ello, como veremos posteriormente en el ejemplo de configuración, estos atributos se definen entre símbolos '*%*'.

* Se ha usado la sintaxis del módulo ConfigParser para Python: http://docs.python.org/library/configparser.html
** La documentación sobre este archivo se puede encontrar en: http://*libvirt.org/formatdomain.html*

- El modo de red debe ser *multicast*, como ya se explico anteriormente.
- Debemos de crear un recurso *graphics* de tipo *VNC* para poder llevar a cabo las tareas de despliegue y gestión, como veremos en apartados posteriores.

Una vez que tenemos todo esto en cuenta, vamos a ver en la *ilustración 23* un ejemplo:

```
<!-- Plantilla base de DVL, entre '%' se encuentran los parámetros modificables -->

<domain type='kvm'>
 <name>%name%</name>
 <uuid>%uuid%</uuid>
 <memory>524288</memory>
 <currentMemory>524288</currentMemory>
 <vcpu>2</vcpu>
 <os>
  <type arch='i686' machine='pc-0.11'>hvm</type>
  <boot dev='cdrom'/>
 </os>
 <features>
  <acpi/>
  <apic/>
  <pae/>
 </features>
 <clock offset='utc'/>
 <on_poweroff>destroy</on_poweroff>
 <on_reboot>restart</on_reboot>
 <on_crash>restart</on_crash>
 <devices>
  <emulator>/usr/bin/kvm</emulator>
  <disk type='file' device='disk'>
   <source file='%file_disk%'/>
   <target dev='hda' bus='ide'/>
  </disk>
  <disk type='file' device='cdrom'>
   <source file='%file_iso%'/>
   <target dev='hdc' bus='ide'/>
   <readonly/>
  </disk>
  <interface type='mcast'>
   <source address='%mcast%' port='%port%'/>
  </interface>
  <serial type='pty'>
   <target port='0'/>
  </serial>
  <console type='pty'>
   <target port='0'/>
  </console>
  <input type='mouse' bus='ps2'/>
  <graphics type='vnc' port='-1' autoport='yes' keymap='es'/>
  <video>
   <model type='cirrus' vram='9216' heads='1'/>
  </video>
 </devices>
</domain>
```

Ilustración 23: Ejemplo de plantilla de máquina virtual.

Como observamos en el ejemplo, el archivo se compone de una serie de recursos, los cuales están debidamente detallados en la documentación de *libvirt*. Este ejemplo sirve perfectamente para la mayoría de los casos de máquinas virtuales que generemos, vamos a comentar brevemente los componentes principales:

- Tipo de dominio, *KVM*, ya que es el la plataforma de virtualización que estamos usando
- El nombre y UUID serán configurados dinámicamente en función del de los atributos correspondientes en el archivo *stage.xml*
- Tamaño de memoria, número de CPUs y arquitectura del procesador, en función de la máquina físicas sobre la que vayamos a desplegar la máquina virtual.
- *File disk* y *File ISO*, rutas al archivo .img e .ISO respectivamente. Serán configurados dinámicamente en función del de los atributos correspondientes en el archivo templates.*xml*
- Tipo de red, *multicast*, como ya se explicó antes. Los atributos *mcast* y *port* serán configurados dinámicamente en función del de los atributos correspondientes en el archivo *stage.xml*
- Elemento *graphics* de tipo *VNC*, de forma que *libvirt* configure un Servidor VNC en la máquina virtual y se puedan llevar a cabo las tareas de despliegue y mantenimiento correctamente.

4.2.5. Configuración de secuencia de arranque de máquinas virtuales

Finalmente, debemos ser capaces de simular la secuencia de arranque de cada máquina virtual, para así poder llevar a cabo las tareas posteriores de configuración y mantenimiento. Para este fin, se debe definir un archivo en el cual se indicará el proceso a seguir. En el momento de lanzar las máquinas virtuales, y con la ayuda de un *script*, se mandarán estas órdenes por VNC. Este proceso será detallado posteriormente, ahora vamos a ver un ejemplo y explicar la sintaxis del archivo.

Este archivo se compondrá de 3 tipos de órdenes:

- *Sleep:* Orden para conseguir un retardo. Su sintaxis es '*sleep k*', donde *k* son el número de segundos.
- *Send Key:* Orden para enviar la pulsación de una tecla en concreto. Su sintaxis es *'send_key tecla'*, donde tecla es un código de tecla.
- *Send String:* Orden para enviar una cadena de caracteres. Su sintaxis es *'send_string cadena'*.

A parte de estas órdenes, podemos declarar valores dinámicos que serán sustituidos posteriormente en función del archivo *stage.xml*. La forma de definir estos valores es la misma que en las plantillas de las máquinas virtuales, entre los valores '*%*'. En la *ilustración 24* podemos observar un ejemplo de este archivo:

```
sleep 5
send_key Return
sleep 70
send_string root
send_key Return
sleep 1
send_string toor
send_key Return
sleep 1
send_string ifconfig eth0 %ip% netmask 255.255.255.0
send_key Return
sleep 1
```

Ilustración 24: Ejemplo de archivo de secuencia de arranque.

4.3. Despliegue

Una vez que nuestra herramienta haya generado el *script* de salida, como resultado de parsear los diferentes archivos de configuración previamente analizados, debe llevarse a cabo el despliegue de todo el escenario definido. Para llevar a cabo esta tarea, vamos a basarnos en el uso de *SSH* y de *VNC*. Primero, vamos a analizar el funcionamiento de cada uno de estos por separado, para después, una vez hayamos entendido el funcionamiento, comentar el *script* de salida que lleva a cabo el despliegue.

4.3.1. SSH

Por *ssh* (*Secure Shell*, intérprete de órdenes segura) se conoce al prótocolo que sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos, y también puede redirigir el tráfico de las X (interfaz gráfica) para poder ejecutar programas gráficos si tenemos un Servidor X corriendo (Daniel J. Barrett,Richard E. Silverman,Robert G. Byrnes, 2005).

Sirviéndonos de esta funcionalidad, y previa configuración de los servidores y clientes de *ssh* en las máquinas del laboratorio – lo cual será explicado en el capítulo adecuado -, seremos capaces de enviar y recibir tráfico entre las distintas máquinas físicas, para así poder crear las instancias de las máquinas virtuales necesarias.

4.3.2. VNC

Si bien *ssh* nos permitirá acceder a cualquier equipo físico del laboratorio de manera remota, queda pendiente acceder a los equipos no físicos, es decir, las máquinas virtuales. Nos surge un problema, ya que al instanciar una máquina virtual, esta arrancará sin ningún servicio corriendo (al trabajar con *Live CDs*, como se está haciendo). Si queremos evitar esto, y poder arrancar la máquina virtual con soporte de *ssh* deberíamos de modificar la imagen del

Sistema Operativo a ejecutar, tarea que aun siendo posible, no es recomendable por 2 motivos:

- Necesidad de *altos conocimientos técnicos*. Modificar un sistema operativo no es una tarea trivial.
- Falta de *escalabilidad*. Aun teniendo los conocimientos técnicos necesarios, uno de los requisitos del proyecto es la escalabilidad, es decir, poder ir incluyendo más Sistemas Operativos en el repertorio de nuestro laboratorio con el menor esfuerzo y repercusión posible. No es viable, en términos de tiempo y esfuerzo, tener que modificar la imagen cada vez que se quiera añadir un nuevo sistema operativo.

Para solucionar este problema, vamos a echar mano del protocolo usado por *VNC*, conocido como *RFB (Remote Frame Buffer),* en concreto de su última versión, la 3.8. (RFB v.3.8., 2009). *RFB* es un sencillo protocolo de acceso remoto a interfaces gráficas de usuario. Funciona en el nivel de *frame-buffer*, lo que significa que se puede aplicar a todos los sistemas gestores de ventanas. En el protocolo *Remote Frame Buffer*, la aplicación que se ejecuta en la máquina donde el usuario opera, es llamado *cliente*. La aplicación que se ejecuta en la máquina donde el *frame-buffer* está localizado (el cual está ejecutando el sistema de ventanas y aplicaciones que el usuario está controlando remotamente) es llamado *servidor*. Por defecto el cliente se conectará al puerto 5900 (y sucesivos, en caso de haber más de una instancia corriendo) del servidor.

Aunque *RFB* comenzó siendo un protocolo relativamente sencillo, se han ido añadiendo nuevas características a medida que han ido saliendo versiones: transferencia de archivos, técnicas más sofisticadas de compresión y seguridad, etc. A pesar de esto, y con el fin de mantener la compatibilidad con las versiones antiguas, el primer paso en cualquier conexión será la correspondiente a la negociación de la versión usada, así como las características de compresión y seguridad. Una vez que la negociación ha tenido éxito, se procederá a mandar las instrucciones conforme a la sintaxis definida en el protocolo.

Un punto importante, y que comentamos previamente en el apartado de definición del entorno, es imprescindible activar el uso de *VNC* mediante la directiva adecuada para que este diseño sea efectivo.

Una vez que entendemos el funcionamiento básico del protocolo, vamos a comentar brevemente el *script* desarrollado – *ilustración 25* -, el cual será el encargado de proveernos la funcionalidad de envío de órdenes a través de *VNC*.

```
import socket
import time
import struct
import sys

class RFBClient:
        keycodes = {
                'BackSpace' : 0xff08,
                # Resto de keycodes
        }

        def __init__(self, address):
                # conectamos
                self.s = s = socket.socket()
                s.connect(address)
```

```
                # negociamos version
                version = s.recv(12)
                print version.strip()
                s.sendall(version)

                # negociamos seguridad
                security_types = ord(s.recv(1))
                for i in range(security_types):
                        s.recv(1)

                s.sendall(chr(1))
                struct.unpack('!I', s.recv(4))

                # iniciamos cliente (compartido)
                s.sendall(chr(1))

        def __send_keycode(self, keycode):
                if (keycode==0xFFFB):
                        self.s.sendall(struct.pack("!BBxxI", 4, 1, 0xffe1))
                        time.sleep(0.05)
                        self.s.sendall(struct.pack("!BBxxI", 4, 1, ord('\'')))
                        time.sleep(0.05)
                        self.s.sendall(struct.pack("!BBxxI", 4, 0, 0xffe1))

                        time.sleep(0.05)

                        self.s.sendall(struct.pack("!BBxxI", 4, 0, ord('\'')))
                        self.s.sendall(struct.pack("!BBxxI", 4, 1, 0xffe1))
                        time.sleep(0.05)
                        self.s.sendall(struct.pack("!BBxxI", 4, 0, 0xffe1))
                        time.sleep(0.05)

                else:
                        self.s.sendall(struct.pack("!BBxxI", 4, 1, keycode))
                        time.sleep(0.05)
                        self.s.sendall(struct.pack("!BBxxI", 4, 0, keycode))


        def send_string(self, string):
                for c in string:
                        self.__send_keycode(ord(c))

        def send_key(self, key):
                self.__send_keycode(self.keycodes[key])

        def parse_command(self, line):
                parsed=line.partition(' ')

                if(parsed[0]=="send_string"):
                        client.send_string(parsed[2])
                elif(parsed[0]=="send_key"):
                        client.send_key(parsed[2].replace('\n', ''));
                elif(parsed[0]=="sleep"):
                        time.sleep(int(parsed[2]))
                else:
                        print "\n\t[!] Comando invalido\n"
```

```
if(len(sys.argv) == 4):
        #abrimos archivo de ordenes
        fd = open(sys.argv[3], "r")

        #conectamos con los datos suministrados
        client = RFBClient((sys.argv[1], int(sys.argv[2])))

        #leemos linea por linea y parseamos el tipo de orden
        fileList = fd.readlines()
        for fileLine in fileList:
                client.parse_command(fileLine)

        #cerramos el descriptor
        fd.close()

elif(len(sys.argv) == 3):
        #conectamos con los datos suministrados
        client = RFBClient((sys.argv[1], int(sys.argv[2])))

        line = sys.stdin.readline()

        while (line!="q\n"):
                client.parse_command(line)
                line = sys.stdin.readline()

else:
        print "\n\t[+] Uso: " + sys.argv[0] + " servidor_VNC puerto archivo_ordenes\n"
```

Ilustración 25: Script envío órdenes por VNC.

Vamos a comentar brevemente el *script*. En primer lugar se importan las bibliotecas necesarias para poder trabajar luego. A continuación se define la clase principal, llamada *RFBclient*, la cual consta de:

- *keycodes*, mediante un diccionario *Python* relacionamos cada tecla con su código. En el código de la figura no se han incluido todos por motivos de espacio.

- método *__init__* (constructor), encargado de conectar con el servidor *RFB*, negociar la versión y la seguridad, e iniciar el envío de órdenes.

- métodos *send_string* y *send_key*, encargados de implementar la funcionalidad de mandar una tecla o una cadena. Usan el método privado *__send_keycode* para ello, el cual construye el paquete con el formato *RFB*.

- Método *parse_command*, encargado de parsear el comando introducido, llamando después al método apropiado dependiendo del tipo de comando.

Finalmente, nos encontramos con el programa principal, que se encarga de validar los parámetros suministrados por consola, crear la instancia de la clase *RFBclient*, y leer el fichero de instrucciones a ejecutar.

4.3.3. Ejemplo de *script* de despliegue

Después de explicar el funcionamiento de *ssh* y el protocolo *RFB* usado en *VNC*, vamos a ver un ejemplo de *script* de despliegue generado por nuestra herramienta, el cual hará uso de ambos componentes.

El *script* es el que se puede observar en la *ilustración 26*, y corresponde al escenario definido previamente en el apartado 4.2.2., no se muestra entero por motivos de espacio. Este código en concreto pertenece al segmento número 2, en el que se definen todos los posibles elementos existentes, por lo que nos sirve perfectamente de ejemplo.

```
ssh root@192.168.1.129 '
cp \/home\/antonio\/Escritorio\/PFC\/firewall.xml
\/home\/antonio\/Escritorio\/PFC\/firewall.xml.tmp;
sed -i "s/%file_iso%/\/home\/antonio\/Escritorio\/PFC\/DVL.iso/g"
\/home\/antonio\/Escritorio\/PFC\/firewall.xml.tmp;
sed -i "s/%name%/gw1/g" \/home\/antonio\/Escritorio\/PFC\/firewall.xml.tmp;
dd if=/dev/zero of=\/home\/antonio\/Escritorio\/PFC\/firewall.xml.img bs=512M
count=4;
sed -i "s/%file_disk%/\/home\/antonio\/Escritorio\/PFC\/firewall.xml.img/g"
\/home\/antonio\/Escritorio\/PFC\/firewall.xml.tmp;
sed -i "s/%uuid%/`uuidgen`/g" \/home\/antonio\/Escritorio\/PFC\/firewall.xml.tmp;
sed -i "s/%mcast%/230.0.0.1/g" \/home\/antonio\/Escritorio\/PFC\/firewall.xml.tmp;
sed -i "s/%port%/8888/g" \/home\/antonio\/Escritorio\/PFC\/firewall.xml.tmp;
sed -i "s/%port2%/8889/g" \/home\/antonio\/Escritorio\/PFC\/firewall.xml.tmp;
virsh -c qemu:///system create \/home\/antonio\/Escritorio\/PFC\/firewall.xml.tmp;
rm \/home\/antonio\/Escritorio\/PFC\/firewall.xml.tmp;

cp \/home\/antonio\/Escritorio\/PFC\/dvl.xml
\/home\/antonio\/Escritorio\/PFC\/dvl.xml.tmp;
sed -i "s/%file_iso%/\/home\/antonio\/Escritorio\/PFC\/DVL.iso/g"
\/home\/antonio\/Escritorio\/PFC\/dvl.xml.tmp;
sed -i "s/%name%/dvl2/g" \/home\/antonio\/Escritorio\/PFC\/dvl.xml.tmp;
dd if=/dev/zero of=\/home\/antonio\/Escritorio\/PFC\/dvl.xml.img bs=512M count=4;
sed -i "s/%file_disk%/\/home\/antonio\/Escritorio\/PFC\/dvl.xml.img/g"
\/home\/antonio\/Escritorio\/PFC\/dvl.xml.tmp;
sed -i "s/%uuid%/`uuidgen`/g" \/home\/antonio\/Escritorio\/PFC\/dvl.xml.tmp;


sed -i "s/%mcast%/230.0.0.1/g" \/home\/antonio\/Escritorio\/PFC\/dvl.xml.tmp;
sed -i "s/%port%/8889/g" \/home\/antonio\/Escritorio\/PFC\/dvl.xml.tmp;
virsh -c qemu:///system create \/home\/antonio\/Escritorio\/PFC\/dvl.xml.tmp;
rm \/home\/antonio\/Escritorio\/PFC\/dvl.xml.tmp;
'

cp /home/antonio/Escritorio/PFC/firewall.boot
/home/antonio/Escritorio/PFC/firewall.boot.tmp;
sed -i "s/%ip%/192.168.2.1/g" /home/antonio/Escritorio/PFC/firewall.boot.tmp;
sed -i "s/%ip2%/192.168.3.1/g" /home/antonio/Escritorio/PFC/firewall.boot.tmp;
python ./vnc_send_keys.py 192.168.1.129 5900
/home/antonio/Escritorio/PFC/firewall.boot.tmp
rm /home/antonio/Escritorio/PFC/firewall.boot.tmp;

cp /home/antonio/Escritorio/PFC/dvl.boot
/home/antonio/Escritorio/PFC/dvl.boot.tmp;
sed -i "s/%ip%/192.168.3.50/g" /home/antonio/Escritorio/PFC/dvl.xml.tmp;
python ./vnc_send_keys.py 192.168.1.129 5901
/home/antonio/Escritorio/PFC/dvl.boot.tmp
rm /home/antonio/Escritorio/PFC/dvl.boot.tmp;

echo -e "send_string route add 'net 192.168.4.0-24 gw 192.168.3.2\nsend_key
Return\nq\n" | python ./vnc_send_keys.py 192.168.1.129 5900
echo -e "send_string iptables -t filter -A INPUT -p tcp --dport 25 -j ACCEPT
\nsend_key Return\nq\n" | python ./vnc_send_keys.py 192.168.1.129 5900
echo -e "send_string iptables -t filter -A INPUT -p tcp --dport 80 -j ACCEPT
\nsend_key Return\nq\n" | python ./vnc_send_keys.py 192.168.1.129 5900
```

```
echo -e "send_string route add 'net 192.168.2.0-24 gw 192.168.3.1\nsend_key
Return\nq\n" | python ./vnc_send_keys.py 192.168.1.129 5901
echo -e "send_string route add 'net 192.168.4.0-24 gw 192.168.3.2\nsend_key
Return\nq\n" | python ./vnc_send_keys.py 192.168.1.129 5901
echo -e "send_string route add default gw 192.168.3.2\nsend_key Return\nq\n" |
python ./vnc_send_keys.py 192.168.1.129 5901

echo -e "send_string start-apache -p 80  -u root \nsend_key Return\nq\n" | python
./vnc_send_keys.py 192.168.1.129 5901
echo -e "send_string start-pgsql -p 80  -u root \nsend_key Return\nq\n" | python
./vnc_send_keys.py 192.168.1.129 5901

echo -e "send_string echo nmap -O 192.168.2.50 | at now + 5minutes\nsend_key
Return\nq\n" | python ./vnc_send_keys.py 192.168.1.129 5901
echo -e "send_string echo ping 192.168.2.50 | at now + 5minutes\nsend_key
Return\nq\n" | python ./vnc_send_keys.py 192.168.1.129 5901
```

Ilustración 26: Script de salida.

Vamos a comentar brevemente este *script:*

- En primer lugar, vemos como mediante *ssh* se conecta a la máquina física con IP 192.168.129 con usuario *root*. A esta máquina se envía una serie de comandos que sirven para configurar los diferentes archivos acorde a los valores de los parámetros definidos en el *stage.xml* y finalmente se levantan las 2 instancias de máquinas virtuales (un *gateway* y un *host*).
- A continuación, haciendo uso del *script* de *VNC*, se configuran los interfaces de red de ambas máquinas virtuales, asignándoles las direcciones IP especificadas.
- El siguiente paso es definir las rutas en ambas máquinas virtuales.
- Finalmente, se cofiguran los filtros para el *firewall*, los servicios y los ataques.

4.4. Gestión

Una vez que hemos configurado el entorno, definido el escenario, y desplegado éste, debemos ser capaces de poder gestionarlo. Para ello, haremos uso del servidor *VNC* desplegado en cada máquina virtual instanciada en los equipos físicos que componen el laboratorio. La forma de conectar previamente a cada equipo será mediante *ssh*, ya que si dejamos directamente abierta la conexión por *VNC* podremos sufrir problemas de *seguridad*, siendo posible que cualquier persona se conecte al escritorio remoto de las diferentes máquinas virtuales. Entraremos a discutir detalladamente estos temas en el apartado siguiente.

En la *ilustración 27* podemos ver como quedaría el esquema final:

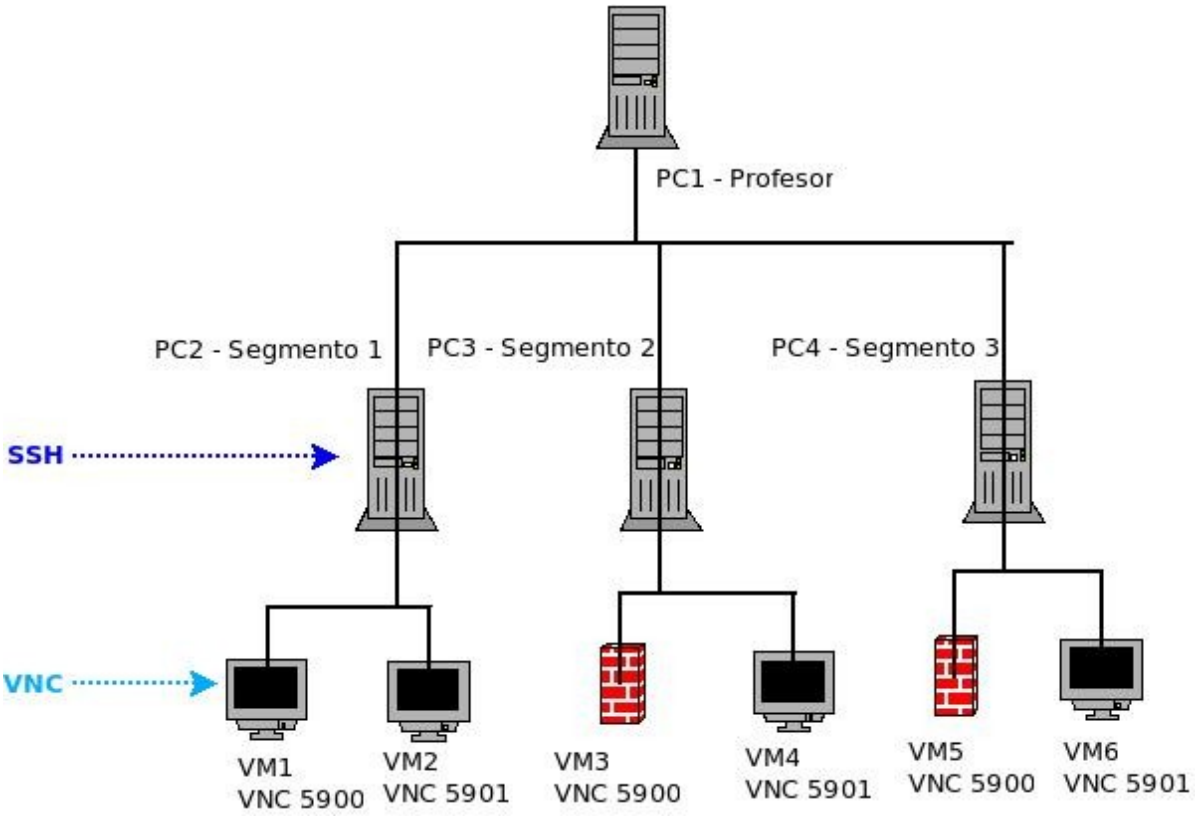

Ilustración 27: Diagrama de conexión por VNC a las máquinas virtuales.

Como podemos observar en la figura, el equipo del profesor se conectaría a cualquiera de los PCs físicos, y una vez dentro, se establecería una conexión al servidor *VNC* correspondiente a la máquina virtual a la que se desee acceder. Para hacer posible este segundo paso, se va a utilizar una técnica conocida como *ssh tunneling*, mediante la cual se encapsula tráfico de otros protocolos, en este caso *VNC,* dentro del tráfico *ssh*. De esta forma, el tráfico *VNC* que envía el profesor irá encapsulado dentro del tráfico *ssh* hasta alcanzar el equipo físico sobre el que está corriendo la máquina virtual, y una vez que llega aquí, el servidor *ssh* será el encargado de conectar con el servidor *VNC* de la máquina virtual correspondiente, encargándose posteriormente de manejar la conexión hasta su finalización.

4.5. Seguridad

Sin duda, uno de los puntos más importantes a la hora de diseñar este proyecto, es la seguridad. Como se analizó en el capítulo anterior al enumerar los requisitos, hay una serie de cuestiones que debemos ser capaces de solventar a la hora de implantar el laboratorio, de forma que la seguridad de este quede completamente cubierta. De no ser así, los efectos negativos podrían ser importantes, ya que se va a tratar con *software* hecho específicamente para auditar la seguridad de sistemas, de forma que un usuario malintencionado podría intentar llegar más allá de lo permitido - el entorno del laboratorio -, y tener acceso a la red Universitaria. A parte de esto, hay otra serie de amenazas que debemos tener en cuenta, y que en caso contrario podrían permitir a los alumnos un uso no autorizado del laboratorio, haciendo que la calificación de estos no sea válida. Por todos estos motivos, queda claro que es imprescindible dedicarle tiempo y esfuerzo a diseñar un laboratorio que sea totalmente seguro.

A continuación, vamos a desgranar en sucesivos apartados las características de seguridad a desplegar. En un primer apartado hablaremos del uso del modo *multicast* de *libvirt*, y como mediante el uso de esta técnica conseguimos hacer transparente el tráfico entre las máquinas virtuales. A continuación hablaremos de como deberán de llevarse a cabo la *autenticación* y las *comunicaciones* entre el ordenador central desde donde trabajarán los profesores, y los equipos físicos sobre los que se desplegará el escenario. Finalmente, discutiremos una serie de medidas que hay que tener en cuenta a la hora de securizar el laboratorio.

4.5.1. *Libvirt* y *multicast*

Cuando se habló sobre la arquitectura de las máquinas virtuales en apartados anteriores, se comentó que dentro de las diferentes opciones que soporta *libvirt* en cuanto a modos de red, se había optado por trabajar con el modo *multicast*. A pesar de las virtudes ya señaladas, debemos verificar que el flujo de información - entre las distintas máquinas que usen este modo de red- sea totalmente transparente a los usuarios finales, principalmente para los alumnos. En caso contrario, éstos podrían interceptar y modificar este tráfico con propósitos no legítimos.

Con el fin de cerciorarnos sobre la transparencia del tráfico generado por *libvirt* en su modo de trabajo en red *multicast*, se ha creado un escenario siguiendo los pasos ya comentados en apartados previos – exactamente como haría cualquier profesor -, y se ha procedido a capturar el tráfico resultante con *Wireshark*[16], una aplicación que nos permite obtener y analizar el tráfico que circula por la red. A continuación, en la *ilustración 28*, presentamos un diagrama con el escenario creado y la máquina donde se captura el tráfico. Como se puede observar en la figura, se han creado dos segmentos de red en sendos equipos físicos diferentes, y desde un tercero se ha procedido a capturar el tráfico generado entre los anteriores.

16 Wireshark (2011), http://www.wireshark.org

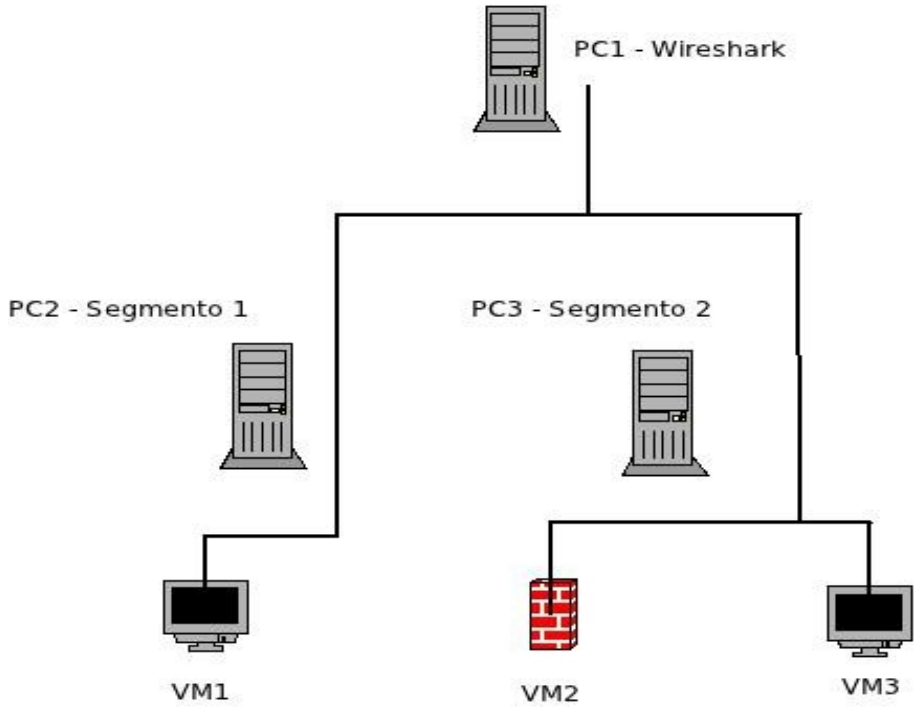

Ilustración 28: Escenario prueba seguridad Multicast de libvirt.

En la *ilustración 29* se muestran los resultados obtenidos con *Wireshark*:

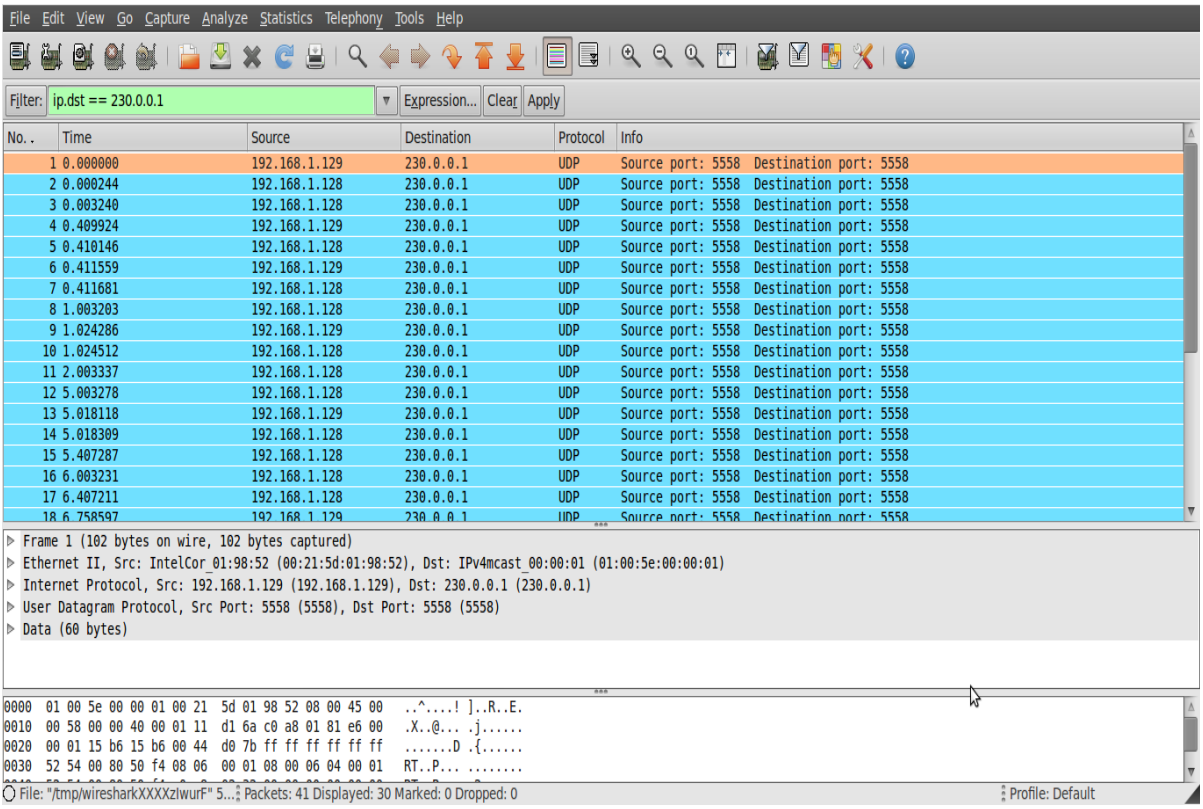

Ilustración 29: Captura de tráfico Multicast de libvirt.

Como podemos observar en la figura, todo el flujo de datos generado por *libvirt* trabajando en modo *multicast* se percibe como tráfico *UDP* desde el exterior. De esta forma, cualquier usuario que tenga acceso al tráfico, verá una serie de tramas *UDP* entre lo que parecen ser puertos aleatorios y no privilegiados, sin aparente sentido. En resumen, usando el modo *multicast* conseguimos el objetivo de que la información transmitida entre las máquinas virtuales con ayuda de *libvirt* sea totalmente transparente.

4.5.2. Autenticación y comunicación segura

Si bien, en el apartado correspondiente al despliegue introducimos *ssh*, no hablamos en ningún momento de sus características en lo referente a la seguridad. Como ya dijimos, *ssh* es un protocolo que hace posible que un cliente (un usuario o un equipo) abra una sesión interactiva en una máquina remota (servidor) para enviar comandos o archivos, además de esto, debemos añadir que este tráfico se hace a través de un *canal seguro*:

- Los datos que circulan entre el cliente y el servidor están cifrados y esto garantiza su confidencialidad (nadie más que el servidor y el cliente pueden leer la información que se envía a través de la red). Como resultado, no es posible controlar la red con un sniffer.

- El cliente y el servidor se autentican uno a otro para asegurarse que las dos máquinas que se comunican son aquellas que las partes creen que son. Un posible atacante ya no podría adoptar la identidad del cliente o de su servidor.

El objetivo de la versión 1 del protocolo, propuesta en 1995, era ofrecer una alternativa a las *shells* existentes, tales como t*elnet*, *rsh*, *rlogin* y *rexec*. Sin embargo, este protocolo tenía un punto débil que permitía a los atacantes introducir datos en los flujos cifrados. Por este motivo, en 1997 se propuso la versión 2 del protocolo (*SSH2*) como un anteproyecto del IETF[17].

En cuanto al funcionamiento de una conexión *ssh*, podemos decir que se realiza en 2 fases, las cuales serán analizadas en sendos apartados:

- En primera instancia, se determina la identidad entre el servidor y el cliente para *establecer un canal seguro* (capa segura de transporte).

- En segunda instancia, el cliente *inicia sesión* en el servidor.

4.5.2.1. Establecimiento de canal seguro

El establecimiento de una capa segura de transporte comienza con la fase de negociación entre el cliente y el servidor para ponerse de acuerdo en los métodos de cifrado que quieren utilizar. El protocolo *SSH* está diseñado para trabajar con un gran número de algoritmos de cifrado, por esto, tanto el cliente como el servidor deben intercambiar primero los algoritmos que admiten.

17 IETF (2005), http://www.ietf.org/html.charters/secsh-charter.html

Después, para establecer una conexión segura, el servidor envía al cliente su clave de *host*. El cliente genera una clave de sesión de 256 bits que cifra con la clave pública del servidor y luego la envía al servidor junto con el algoritmo utilizado. El servidor descifra la clave de sesión con su clave privada y envía al cliente un mensaje de confirmación cifrado con la clave se sesión. Después de esto, las comunicaciones restantes se cifran gracias a un algoritmo de cifrado simétrico, mediante la clave de sesión compartida entre el cliente y el servidor.

La seguridad de la transacción se basa en la confianza del cliente y el servidor en que las claves *host* de cada una de las partes son válidas. Así, cuando se conecta por primera vez con el servidor, el cliente muestra generalmente un mensaje en el que le pide que acepte la comunicación, mostrándole un *hash* de la clave *host* del servidor:

Host key not found from the list of known hosts. Are you sure you want to continue connecting (yes/no)?

Si el usuario valida la conexión, el cliente guarda la clave *host* del servidor para evitar tener que repetir esta fase. Por el contrario, dependiendo de su configuración, el servidor puede, a veces, verificar que el cliente es quien dice ser. Si el servidor tiene un lista de *host* autorizados para la conexión, cifrará el mensaje utilizando la clave pública del cliente (que se encuentra en la base de datos de claves del *host*) para verificar si el cliente es capaz de descifrarla con su clave privada, esto se llama *challenge* (IETF, 2006).

Como vemos, el establecimiento de la conexión a través de un canal seguro, hará que todo el flujo de información transmitida a partir de ese momento entre el equipo del profesor y los demás, se haga de una forma totalmente segura, cubriendo así las necesidades requeridas.

4.5.2.2. Inicio de sesión

Una vez que se ha establecido la conexión segura entre el cliente y el servidor, el cliente debe conectarse al servidor para obtener un derecho de acceso. Existen diversos métodos (IETF, 2006):

- El método más conocido es la contraseña tradicional. El cliente envía un nombre de acceso y una contraseña al servidor a través de la conexión segura y el servidor verifica que el usuario en cuestión tiene acceso al equipo y que la contraseña suministrada es válida.

- Un método menos conocido, pero más flexible, es el uso de claves públicas. Si el cliente elige la clave de autenticación, el servidor creará un *challenge* y le dará acceso al cliente si éste es capaz de descifrar el *challenge* con su clave privada. Este será el método elegido a la hora de realizar la implantación del laboratorio.

4.5.3. Otras medidas de seguridad

En este apartado, se va a presentar una serie de medidas que ayudarán a elevar el grado de seguridad del laboratorio, evitando posibles usos no legítimos de este entorno, tanto por parte de usuarios externos, como de los propios alumnos:

- *Filtrado de tráfico entrante y saliente*: se debe de filtrar el flujo de datos que entra y sale de la red del laboratorio, permitiendo sólo el tráfico que se considere necesario y desechando el restante. Para esto, se propone el uso de un *firewall* como *Iptables*[18], que nos ayudará en esta tarea.

- *Monitorización de roles de los alumnos:* se debe poder controlar que los alumnos sólo ejerzan un rol simultaneamente, en caso contrario los resultados de la evaluación se podrían ver manipulados. Para esto, se propone el uso de certificados a nivel *ssh*, que permitirán el acceso a las máquinas físicas para fortificarlas a los alumnos destinados a ello, e impidiendo esta tarea a los demás.

- *Separación de las máquinas*: se recomienda usar máquinas físicas diferentes a la hora de desplegar el escenario, con respecto a las usadas por los alumnos para llevar a cabo las prácticas. Con esto ganamos un nivel más de seguridad, al no tener los alumnos acceso físico a las máquinas sobre las que está montado el escenario.

- *Software actualizado:* ya que se va a usar *software* muy específico, se recomienda estar al tanto de actualizaciones de seguridad que puedan surgir, para así evitar posibles brechas se seguridad.

4.6. Aplicación desarrollada

Finalmente, vamos a hablar sobre el diseño de la aplicación encargada de leer los distintos archivos de configuración generados, para parsearlos, comprobar su validez, y generar automáticamente el *script* que hará posible el despliegue del escenario definido a lo largo de la red del laboratorio.

Para llevar a cabo esta aplicación, se ha optado por el uso del lenguaje *Python*[19], al igual que con el *script* de envío de comandos por *VNC*. La razón de esta elección ha sido la sencillez, la potencia, y la gran cantidad de bibliotecas disponibles para afrontar cualquier tipo de tarea.

En este capítulo vamos, en primer lugar, a comentar las diferentes *bibliotecas* usadas a lo largo del programa, esto nos permitirá entender mejor cada uno de las distintas partes de la aplicación. Una vez hecho esto, se pasará a presentar las partes más importantes del código fuente y que son las encargadas de proporcionar la funcionalidad final.

---

18 Iptables (2010), http://www.netfilter.org/

19 Python (2011), http://www.python.org/

4.6.1. Bibliotecas

Una de las principales razones por la que se eligió *Python* a la hora de desarrollar el código de este proyecto, es el amplio abanico de bibliotecas existentes (Python, 2008). A continuación vamos a presentar las principales bibliotecas usadas, comentándolas brevemente, esto nos permitirá comprender mejor el grueso del código fuente, sobre el que hablaremos posteriormente.

- Biblioteca *sys*[20], parámetros y funciones específicas del sistema. Este módulo provee acceso a una serie de variables usadas o mantenidas por el intérprete, y a funciones que interactuan con el intérprete estrechamente. En nuestro caso será usada para manejar la finalización del programa.

- Biblioteca *optparse*[21], manejo de parámetros. El módulo *optparse* es una alternativa moderna para gestionar los parámetros suministrados por la línea de comando. Ofrece varias funciones no disponibles en bibliotecas más antiguas como *getopt*, incluyendo la conversión de tipo, *callbacks* opcionales, y la generación de ayuda automática. Lo usaremos para identificar los argumentos pasados al programa.

- Biblioteca *re*[22], expresiones regulares. El módulo *re* contiene diversas funciones para trabajar con expresiones regulares, como puede ser: buscar patrones dentro de una cadena (*search*), comprobar si una cadena se ajusta a un determinado criterio descrito mediante un patrón (*match*), dividir la cadena usando las ocurrencias del patrón como puntos de ruptura (*split*), sustituir todas las ocurrencias del patrón por otra cadena (sub), etc. En nuestro caso será usada para esta última tarea, sustitución de cadenas dentro de otras.

- Biblioteca *lxml*[23], *XML* y *HTML* con Python. El *toolkit* de *XML lxml* es un *binding* de las bibliotecas de C *libxml2* y *libxslt*. Combina la velocidad y las características de estas bibliotecas con la simplicidad de una *API* nativa de *Python*, en su mayoría compatible, pero superior a la *API ElementTree*. Será usada para llevar a cabo el parseo de los distintos archivos *XML* de configuración.

- Biblioteca *configparser*[24], parseo de archivos de configuración. ConfigParser es un módulo muy útil a la hora de leer y escribir archivos de configuración al estilo de los *.ini* de Windows, con distintas secciones delimitadas por un nombre de sección entre corchetes y pares clave-valor con la forma "clave: valor" o "clave=valor". Lo usaremos para leer los diferentes parámetros de configuración establecidos por el usuario.

---

20 Biblioteca *sys*, http://docs.python.org/library/sys.html

21 Biblioteca *optparse*, http://docs.python.org/library/optparse.html

22 Biblioteca *re*, http://docs.python.org/library/re.html

23 Biblioteca *lxml,* http://lxml.de/

24 Biblioteca *configparser*, http://docs.python.org/library/configparser.html

4.6.2. Principales componentes

Después de haber introducido las diferentes bibliotecas usadas en el proyecto, vamos a comentar las principales secciones de código, esto nos servirá para ver como se han desarrollado los diagramas de actividad presentados en el capítulo anterior y comprender el diseño del código.

- Manejo de parámetros. El programa sólo requiere un parámetro, la ruta al archivo de configuración, en caso de que no se le facilite, intentará leer por defecto el archivo *'config.cfg'*. A continuación, en la *ilustración 30*, presentamos el extracto de codigo que se encarga de esta tarea:

```
parser = OptionParser(version="%prog 1.0")

parser.add_option("-f", "--file", action="store", type="string", dest="filename",  help="CONFIGURATION file [default:
                                                                                    %default]", default="./config.cfg")

(options, args) = parser.parse_args()

if (len(args) != 0):
        parser.error("wrong number of arguments")
```

Ilustración 30: Manejo de parámetros.

- Parseo de archivos de configuración. Los profesores definirán en el archivo de configuración las rutas necesarias para que la aplicación funcione correctamente, por lo que debemos de leer este archivo y parsearlo para leer los valores de estas rutas. A continuación, en la *ilustración 31*, presentamos el extracto de codigo que se encarga de esta tarea:

```
try:
        open(options.filename)

        try:
                config_file = ConfigParser.SafeConfigParser()                   #Inicializamos el ConfigParser
                config_file.read(options.filename)                              #Leemos el archivo de configuracion

                vnc_sendkeys = config_file.get('main', 'vnc_sendkeys')
                nput_stage = config_file.get('main', 'input_stage')
                templates = config_file.get('main', 'templates')
                output_script = config_file.get('main', 'output_script')


        except Exception, e:
        print "\n\n\t[+] An unexpected exception was encountered while parsing CONFIG file. %s\n\n" % str(e)
        sys.exit(1)

except IOError, e:
        print("({0})".format(e))
        sys.exit(1)
```

Ilustración 31: Parseo de archivos de configuración

- Definición de estructuras de datos. Una parte fundamental, es definir las estructuras de datos que nos servirán para trabajar con los datos proporcionados. Definiremos una clase para cada uno de los elementos que comprenden el escenario, tal y como se definió en el análisis del sistema. Estas clases tendrán los atributos necesarios, y una serie de métodos que permitirán configurarlas, gestionarlas e imprimir por pantalla su estado. A continuación, en la *ilustración 32*, presentamos el extracto de codigo que se encarga de esta tarea, no se han incluido todas las clases por motivo de espacio:

```
class Segment:
        '''Clase que implementa un segmento de red'''
        def __init__(self, attrs):
                try:
                        self.label = attrs['label']
                        self.host = attrs['host']
                        self.mcast = attrs['mcast']
                        self.port = attrs['port']
                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing <SEGMENT> "
                sys.exit(1)

                self.hosts = []
                self.gateways = []

        def add_host(self, host):
                self.hosts.append(host)

        def add_gateway(self, gateway):
                self.gateways.append(gateway)

        def get_port(self):
                return self.port

        def print_segment(self):
                print "SEGMENT:\n", self.label, " ", self.host, " ", self.mcast, " ", self.port
                for gateway in self.gateways:
                        gateway.print_gateway()
                for host in self.hosts:
                        host.print_host()

        def configure_segment(self, fd, vnc_sendkeys, mcast_link_port):
                try:
                        script_line = '\nssh root@%s \'\n' % self.host

                        fd.write(script_line)

                        for gateway in self.gateways:
                                gateway.create_gw(fd, self.mcast, mcast_link_port, self.port)
                        for host in self.hosts:
                                host.create_host(fd, self.mcast, self.port)

                        fd.write('\'\n\n')

                        for gateway in self.gateways:
                                gateway.configure_gw( fd,vnc_sendkeys, self.host)
                        for host in self.hosts:
                                host.configure_host( fd, vnc_sendkeys, self.host)

                        for gateway in self.gateways:
                                gateway.configure_routes( fd, vnc_sendkeys, self.host)
                                gateway.configure_ipchains(fd, vnc_sendkeys, self.host)
                        for host in self.hosts:
                                host.configure_routes(fd, vnc_sendkeys, self.host)
                                host.configure_services(fd, vnc_sendkeys, self.host)
                                host.configure_events(fd, vnc_sendkeys, self.host)

                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s file" % fd
```

Ilustración 32: Definición de estructuras de datos.

- Parseo de archivox *XML* de configuración. Una vez que tenemos definidas las estructuras de datos adecuadas, debemos de parsear los archivos *XML*, validarlos y guardar en éstas los valores leidos. A la hora de parsear los archivos, debemos de hacerlo recursivamente, e ir comprobando los parámetros de los sucesivos *tags* que vayamos leyendo. A continuación, en la *ilustración 33*, presentamos el extracto de codigo que se encarga de parsear el archivo *templates.xml*:

```
class Template_parser:
'''Clase que parsea y valida la definicion de una Plantilla'''

def __init__(self, template_path):
        self.template_path = template_path

        try:
                self.doc = etree.parse (self.template_path)    #Abrimos archivo de escenario
        except Exception, e:
        print "\n\n\t[+] An unexpected exception was encountered while parsing XML file: %s\n\n" % str(e)
        sys.exit(1)

#def __del__(self):

def __parse_vm(self, root):
        if (not(root.attrib.get('name') and root.attrib.get('path') and root.attrib.get('template') and
root.attrib.get('boot'))):
                print "\n\n\t[+] Error: Vm tag has no NAME or PATH or TEMPLATE or BOOT
                                                                parameters\n\n"
                sys.exit(1)
        for child in root:
                if (child.tag == 'service'):
                        gateway = self.__parse_service(child)
                elif (child.tag == 'attack'):
                        host = self.__parse_attack(child)
                else:
                        print "\n\n\t[+] Error: Tag element different from <SERVICE> or
<ATTACK>\n\n"
                        sys.exit(1)

def __parse_service(self, root):
        if (not(root.attrib.get('name') and root.attrib.get('init'))):
                print "\n\n\t[+] Error: Service tag has no NAME or INIT parameters\n\n"
                sys.exit(1)

        for child in root:
                if (child.tag == 'param'):
                        gateway = self.__parse_param(child)
                else:
                        print "\n\n\t[+] Error: Tag element different from <PARAM>\n\n"
                        sys.exit(1)

def __parse_attack(self, root):
        if (not(root.attrib.get('name') and root.attrib.get('command'))):
                print "\n\n\t[+] Error: Attack tag has no NAME or COMMAND parameters\n\n"
                sys.exit(1)

        for child in root:
                if (child.tag == 'type'):
                        gateway = self.__parse_type(child)
                else:
                        print "\n\n\t[+] Error: Tag element different from <TYPE> \n\n"
                        sys.exit(1)

def __parse_param(self, root):
                if (not(root.attrib.get('name') and root.attrib.get('mod'))):
                        print "\n\n\t[+] Error: Param tag has no NAME or MOD(empty?-> use " ")
parameters\n\n"
                        sys.exit(1)
```

```
def __parse_type(self, root):
                if (not(root.attrib.get('name'))):
                        print "\n\n\t[+] Error: Type tag has no NAME parameter\n\n"
                        sys.exit(1)



                for child in root:
                        if (child.tag == 'param'):
                                gateway = self.__parse_param(child)
                        else:
                                print "\n\n\t[+] Error: Tag element different from <PARAM>\n\n"
                                sys.exit(1)

        def parse_template(self):
                root = self.doc.getroot()                       #Cogemos el elemento raiz del escenario

                if (root.tag != 'templates'):
                        print "\n\n\t[+] Error: Root element different from <TEMPLATES>\n\n"
                        sys.exit(1)

                for child in root:
                        if (child.tag == 'vm'):
                                segment = self.__parse_vm(child)
                        else:
                                print "\n\n\t[+] Error: Tag element different from <VM>\n\n"
                                sys.exit(1)
```

Ilustración 33: Parseo de archivos XML de configuración.

- Generación del *script* de salida. Después de parsear el escenario, y verificar su validez, es necesario generar el *script* de salida. Esto se hace a partir de las estructuras de datos creadas previamente. A continuación, en la *ilustración 34*, presentamos el extracto de codigo que se encarga de esta tarea, no se han incluido todos los métodos por motivo de espacio:

```
def configure_segment(self, fd, vnc_sendkeys, mcast_link_port):
                try:
                        script_line = '\nssh root@%s \'\n' % self.host

                        fd.write(script_line)

                        for gateway in self.gateways:
                                gateway.create_gw(fd, self.mcast, mcast_link_port, self.port)
                        for host in self.hosts:
                                host.create_host(fd, self.mcast, self.port)

                        fd.write('\'\n\n')

                        for gateway in self.gateways:
                                gateway.configure_gw( fd,vnc_sendkeys, self.host)
                        for host in self.hosts:
                                host.configure_host( fd, vnc_sendkeys, self.host)

                        for gateway in self.gateways:
                                gateway.configure_routes( fd, vnc_sendkeys, self.host)
                                gateway.configure_ipchains(fd, vnc_sendkeys, self.host)
                        for host in self.hosts:
                                host.configure_routes(fd, vnc_sendkeys, self.host)
                                host.configure_services(fd, vnc_sendkeys, self.host)
                                host.configure_events(fd, vnc_sendkeys, self.host)

                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s file" % fd
```

```
....

        def create_host(self, fd, mcast, port):
                try:
                        filtered_path = self.filter_string(self.path)
                        filtered_template = self.filter_string(self.template)

                        script_line = 'cp %s %s.tmp;\n' % (filtered_template, filtered_template)
                        script_line += 'sed -i \"s/%%file_iso%%/%s/g\" %s.tmp;\n' % (filtered_path,
filtered_template)
                        script_line += 'sed -i \"s/%%name%%/%s/g\" %s.tmp;\n' % (self.name, filtered_template)
                        script_line += 'dd if=/dev/zero of=%s.img bs=512M count=4;\n' % filtered_template


                        script_line += 'sed -i \"s/%%file_disk%%/%s.img/g\" %s.tmp;\n' % (filtered_template,

        filtered_template)
                        script_line += 'sed -i \"s/%%uuid%%/`uuidgen`/g\" %s.tmp;\n' % filtered_template
                        script_line += 'sed -i \"s/%%mcast%%/%s/g\" %s.tmp;\n' % (mcast, filtered_template)
                        script_line += 'sed -i \"s/%%port%%/%s/g\" %s.tmp;\n' % (port, filtered_template)
                        script_line += 'virsh -c qemu:///system create %s.tmp;\n' % filtered_template
                        script_line += 'rm %s.tmp;\n' % filtered_template
                        fd.write(script_line)
                        #otras confs

                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s file" % fd
```

Ilustración 34: Generación del script de salida.

5

IMPLEMENTACIÓN

En este capítulo vamos a detallar la forma de llevar a cabo la implementación del proyecto en la red del laboratorio. Una vez que hemos analizado y diseñado la arquitectura, así como los distintos componentes que forman parte de esta, debemos extrapolarlo a un entorno físico. La implantación será llevada a cabo sobre un laboratorio que cumpla los requisitos *hardware* presentados en el capítulo 3. Se supondrá que estos equipos estarán completamente formateados, sin ningún tipo de *software* instalado en ellos, por lo que se detallará desde la instalación del sistema operativo, pasando por la configuración de las herramientas usadas, hasta la securización final del entorno. Se ha decidido hacerlo de esta forma - y no basándonos en configuraciones previas que pudieran existir en las diferentes aulas -, para que el despliegue en un laboratorio pueda ser llevado a cabo por cualquier profesor siguiendo los pasos que se indicarán posteriormente, independientemente de que nunca lo haya hecho previamente o de la amalgama de laboratorios totalmente heterogéneos con la que se pueda encontrar.

Para hacer más fácil, modular, y comprensible el despliegue, se va a realizar una división en fases, cada una de las cuales tendrá correspondencia con su respectivo apartado dentro de este capítulo.

En el primero de estos apartados, vamos a hablar sobre la *instalación del software* necesario en los distintos equipos físicos. Esto incluirá los componentes que soportan la virtualización, el despliegue de los escenarios, las imágenes de los sistemas operativos a usar, las herramientas para la gestión de los escenarios deplegados, y la aplicación desarrollada para el proyecto.

En el siguiente apartado, se indicará detalladamente como *configurar* cada uno de los elementos previamente instalados, de esta forma conseguiremos que todo el *software* trabaje conjuntamente y conforme al diseño realizado.

Finalemente, dedicaremos un apartado para encargarnos de la *securización* del entorno desplegado. Se procederá a aplicar las medidas comentadas en el capítulo anterior para cubrir todos los requisitos de seguridad expuestas, y que de esta forma el laboratorio sea apto para su despliegue en un entorno universitario.

5.1. Instalación del *software* necesario

En este apartado se va a detallar la instalación de los distintos componentes *software* necesarios. Por motivos de claridad se hará en apartados distintos, en los que hablaremos de la instalación de: la herramienta que soportará la *virtualización* y sus respectivos complementos, la herramienta para poder realizar el *despliegue*, las *imágenes de los sistemas operativos* a virtualizar, las herramientas de *gestión* usadas, y finalmente la propia *aplicación desarrollada* en este proyecto. Para complementar todos estos apartados, en el Anexo B se encuentra detallada la instalación del sistema operativo Ubuntu, que soporta el resto del *software* usado.

5.1.1. Virtualización

Una vez que hemos instalado el sistema operativo en todos los equipos del laboratorio, vamos a proceder a instalar el *software* que nos permitirá usar las técnicas de virtualización, así como una serie de complementos que nos facilitarán la creación y el mantenimiento de los escenarios.

Antes de proceder a la instalación, debemos verificar que los equipos cumplen con los prerequisitos necesarios, esto es, soporte de virtualización en la CPU. Para ello, dependiento de la arquitectura del procesador - Intel ó AMD -, debemos de ejecutar los comandos que observamos en las *ilustraciones 35 y 36*.

Ilustración 35: Verificación soporte virtualización en la CPU, arquitectura AMD.

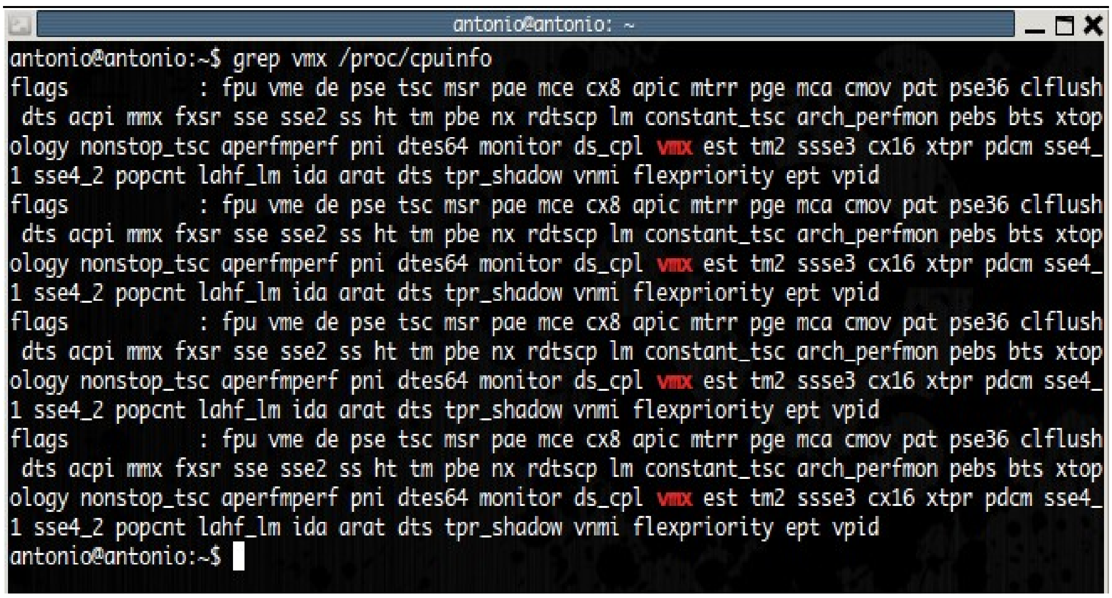

Ilustración 36: Verificación soporte virtualización en la CPU, arquitectura Intel.

En la primera de ellas, arquitectura AMD, vemos como la salida es vacía, mientras que en la segunda, arquitectura Intel, vemos como el procesador si que soporta virtualización. Al menos uno de los dos comandos debe devolvernos una salida por pantalla similar a la mostrada en la última figura, en caso contrario el procesador no dispone de soporte para virtualizar, por lo que no nos sería útil en este proyecto.

Una vez hecha esta verificación, procederemos a instalar el *software* necesario, como podemos observar en la *ilustración 37*.

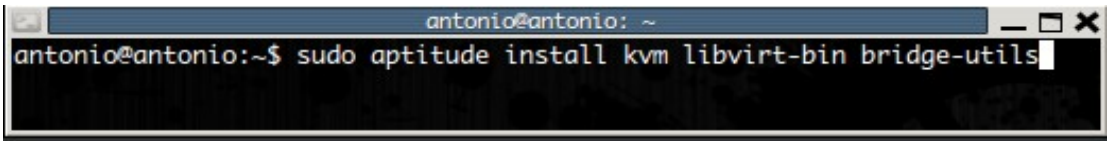

Ilustración 37: Instalación del software de virtualización.

Vamos a explicar brevemente los paquetes instalados:

- *Kvm: Software* de virtualización.
- *Libvirt-bin:* Aplicaciones que usan la biblioteca *libvirt* como intermediario entre *kvm* a la hora de virtualizar.
- *Bridge-utils:* Conjunto de utilidades para trabajar con redes en Linux.

Con esto ya tendríamos instalado y listo para utilizarse todo el *software* relativo a la virtualización.

5.1.2. Despliegue

El siguiente paso, una vez hecho lo propio con las herramientas que soportarán la virtualización, es instalar el *software* encargado de desplegar los escenarios a lo largo de la red del laboratorio. Para ello usaremos *openssh-server* y *openssh-client*, implementación libre - del servidor y cliente respectivamente – del protocolo *ssh*.

Para llevar a cabo esta tarea, debemos de instalar los paquetes tal y como aparece en la *ilustración 38*.

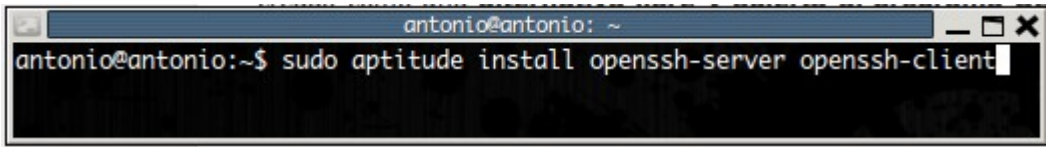

Ilustración 38: Instalación del software de virtualización.

5.1.3. Imágenes de Sistemas Operativos

Los profesores podrán elegir entre una serie de sistemas operativos a desplegar por los escenarios, para que esto sea posible, la aplicación debe tener acceso a una serie de imágenes de los sistemas operativos deseados en formato *iso*[25]. Dependiendo de los sistemas operativos que se vayan a usar, se deberá proceder a descargar uno u otro, por lo que a continuación ofrecemos el enlace a un listado en el que se muestran algunas de las cientos de distribuciones existentes, y que puede servir de guía a la hora de elegir:

http://en.wikipedia.org/wiki/Comparison_of_Linux_distributions#General

5.1.4. Gestión

A la hora de realizar la gestión del laboratorio, y del escenario desplegado en él, vamos a hacer uso de una herramienta con interfaz gráfica (*Graphic User Interface*, Interfaz de Usuario Gráfica). Mediante esta herramienta, se hará más cómoda y eficiente la tarea de mantenimiento. El *software* a usar, como ya se comentó previamente, es *Virt Manager*, que será instalado tal y como se puede observar en la *ilustración 39*.

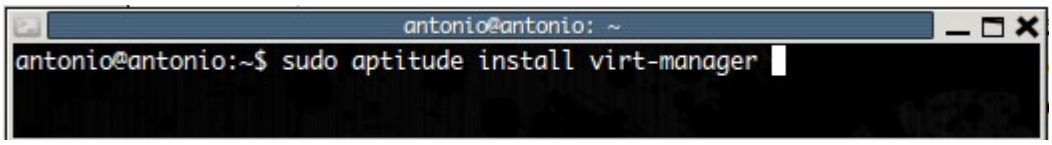

Ilustración 39: Instalación del software de gestión.

5.1.5. Aplicación desarrollada

Finalmente, nos debemos ocupar de la aplicación realizada para el proyecto. Se han desarrollado dos *scripts*: programa principal y programa para envío de comandos mediante *VNC*. Dado que ambos han sido hechos en *Python*, no debemos realizar ninguna instalación, ya que estamos hablando de un lenguaje interpretado. Un lenguaje de programación es diferente al lenguaje máquina, por lo tanto, debe traducirse para que el procesador pueda comprenderlo. Un programa escrito en un lenguaje interpretado requiere de un programa auxiliar (el intérprete), que traduce los comandos de los programas según sea necesario (Stallings, W., 2008).

Teniendo esto en cuenta, simplemente necesitaremos un intérprete, en este caso un intérprete de *Python,* para hacer correr nuestros dos *scripts*. Y dado que Ubuntu 10.04 viene equipado por defecto con este intérprete, no será necesario llevar a cabo instalación ninguna.

25 ISO9960 Specification (1995), http://users.telenet.be/it3.consultants.bvba/handouts/ISO9960.html

5.2. Configuración

Una vez realizada la instalación de todo el *software* necesario, se va a llevar a cabo la configuración de este. Esta configuración será explicada en diferentes apartados: en el primero de ellos discutiremos los pasos necesarios para poner a punto el *software* de *virtualización*. Posteriormente nos ocuparemos de configurar el *software* utilizado en el *despliegue*. En un tercer apartado nos centraremos en la configuración de las *imágenes de los sistemas operativos*. Y finalemente, hablaremos de la *aplicación desarrollada* para este proyecto y como configurarla adecuadamente.

5.2.1. Virtualización

Por defecto, el demonio *libvirtd* se encarga de que *kvm* sólo acepte conexiones locales. Para cambiar este comportamiento debemos de editar el archivo '/etc/libvirt/qemu.conf' y configurar mediante la variable *vnc_listen* que acepte conexiones de cualquier interfaz, como observamos en la *ilustración 40*.

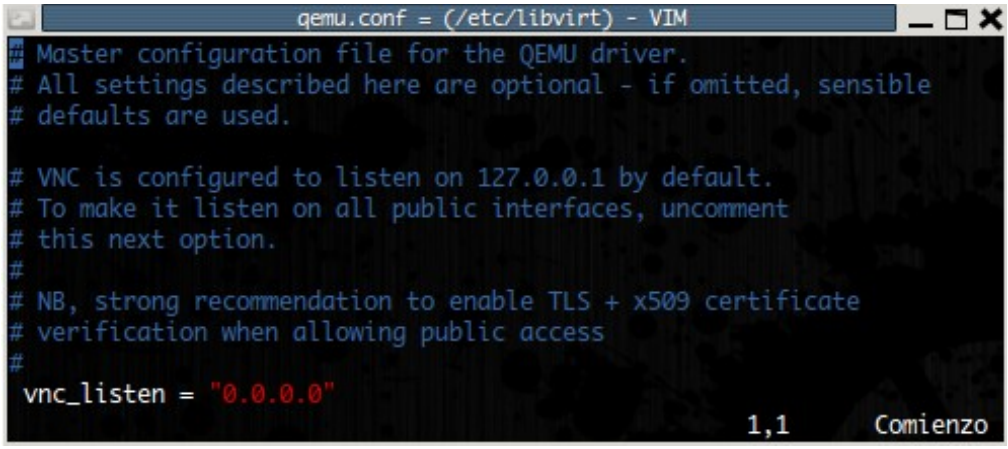

Ilustración 40: Configuración del archivo 'qemu.conf'.

Para que este cambio surta efecto, debemos reiniciar el demonio, tal y como se observa en la *ilustración 41*.

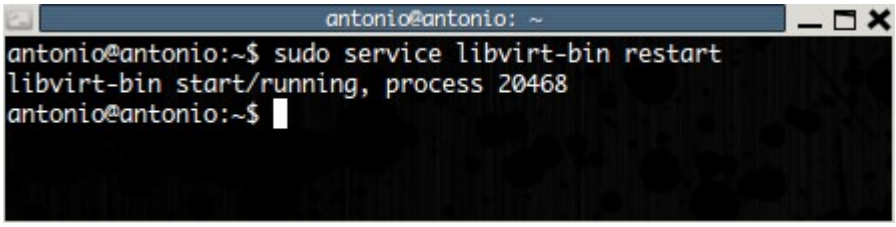

Ilustración 41: Reinicio del demonio libvirt.

A pesar de haber configurado el servicio correctamente, debemos recordar - ya fue explicado en el capítulo anterior - que a la hora de crear una máquina virtual, el archivo de

configuración *xml* usado por *libvirt*, tiene que contener una sección en la que se le añada un elemento *graphics*, de tipo *vnc*. En la figura 5.8. observamos un ejemplo, en el cual declaramos este tipo de recurso, indicándolo que le asigne el puerto *VNC* automáticamente al darle el valor '-1' al parámetro '*port*'.

```
<input type='mouse' bus='ps2'/>
<graphics type='vnc' port='-1' autoport='yes' keymap='es'/>
<video>
  <model type='cirrus' vram='9216' heads='1'/>
```

Ilustración 42: Ejemplo de elemento graphics en archivo xml de máquina virtual.

Figura 5.8.

Después de esto, ya se está en disposición de conectarse remotamente, mediante *VNC,* a cualquier máquina virtual que haya sido creada según lo indicado.

5.2.2. Despliegue

El siguiente paso es configurar el *software* de despliegue, *ssh*, para poder realizar a través de este el despliegue de los escenarios en la red del laboratorio. En este apartado veremos brevemente la forma de configurar el servidor *ssh*, ya que en apartados posteriores se analizará en detalle su configuración debido a las implicaciones con respecto a la seguridad que tiene esta herramienta.

El archivo de configuración principal es '*/etc/ssh/ssh_config*', contiene una serie de parámetros configurables - no aparecen todos, una lista completa la podemos encontrar en la sección 5 del *man*[26] -, los cuales tienen un valor por defecto, que en caso de querer modificarlo se deberá descomentar y darle el nuevo valor deseado.

Una vez realizado los cambios deseados, será necesario reiniciar el servicio para que estos tengan efecto.

5.2.3. Imágenes de Sistemas Operativos

Una vez que se han descargado las imágenes, es conveniente - aunque no necesario - moverlas a una carpeta común, para así tenerlas organizadas y seguir una pauta de cara al futuro.

En este caso, se ha decidido crear una carpeta llamada '*ISOs*', en la cual están todas las imagenes descargadas. A modo de ejemplo, sólo se ha descargado una imagen, correspondiente al sistema operativo *Damn Vulnerable Linux*[27], la cual hemos alojado dentro de la carpeta creada con el nombre '*DVL.iso*'.

---

26 *Man* de ssh, sección 5. http://linux.die.net/man/5/ssh_config

27 Damn Vulnerable Linux (2011), http://www.damnvulnerablelinux.org/

5.2.4. Aplicación desarrollada

Finalmente, nos debemos ocupar de la aplicación desarrollada, y de los distintos archivos de configuración que forman parte de ella.

El primer paso, al igual que se ha hecho con las imágenes de los sistemas operativos, es crear una carpeta llamada '*PFC*', donde se encontrarán los siguientes elementos:

- *templates.xml*
- *vnc_send_keys.py*
- *pfc.py*

A continuación, crearemos una carpeta por cada sistema operativo que vayamos a poner a disposición de los profesores, en este caso sólo una, la cual llamaremos '*DVL*'. Esta carpeta estará compuesta de los siguientes archivos:

- *dvl.xml*
- *dvl.boot*

Una vez creada la estructura de directorios correctamente, vamos a pasar a configurar los archivos necesarios: *templates.xml*, *dvl.xml* y *dvl.boot*.

5.2.4.1. Archivo *templates.xml*

El contenido de este archivo ya ha sido explicado anteriormente, por lo que ahora vamos a centrarnos simplemente en el *tag* donde definimos las rutas del resto de componentes, la cual debe de concordar con la estructura creada recientemente.

En la *ilustración 43* podemos observar como quedaría.

```
<vm name="dvl" path="/home/antonio/Escritorio/ISOs/DVL.iso" template="/home/antonio/Escritorio/DVL/dvl.xml"
        boot="/home/antonio/Escritorio/DVL/dvl.boot">

        ....
</vm>
```

Ilustración 43: Configuración de rutas en el archivo templates.xml.

5.2.4.2. Archivo *dvl.xml*

El contenido de este archivo de configuración ya ha sido explicado previamente, la configuración final sería la mostrada en la *ilustración 24*.

5.2.4.3. Archivo *dvl.boot*

El contenido de este archivo de configuración ya ha sido explicado previamente, la configuración final sería la mostrada en la *ilustración 25*.

5.3. Seguridad

En este último apartado, se van a comentar las medidas de seguridad necesarias - tal y como se explicó en el capítulo anterior - para que el laboratorio desplegado tengo un alto nivel de seguridad. El esquema seguido a la hora de aplicar estas medidas será el presentado en el capítulo previo, apartado 4.5., donde se han desgranado las medidas necesarias a tomar en cuanto a seguridad.

En primer lugar, se dedicará un apartado a configurar la tupla *servidor-cliente de ssh* de forma que trabajen de la manera más segura posible. Para conseguir esto, se deberán realizar una serie de modificaciones en los archivos de configuración correspondientes, además de la creación de los certificados a usar.

A continuación, y siguiendo el esquema ya indicado, dedicaremos otro apartado para explicar una *serie de prácticas* que ayudarán a fortalecer la seguridad del laboratorio implantado.

5.3.1. Servidor-Cliente *SSH*

Openssh nos permite un grado alto de configuración, por lo que vamos a aprovecharlo para tomar una serie de medidas que mejorarán ampliamente la seguridad en las comunicaciones hechas entre los equipos del laboratorio. En cada apartado se manejará el concepto de servidor y cliente *ssh*, en el primer caso se hace en referencia a todos los equipos del laboratorio que poseen servidores *ssh* (por lo que las medidas indicadas se deberán realizar en todos ellos), mientras que en el segundo caso se hace en referencia al equipo manejado únicamente por los profesores (por lo que las medidas indicadas se deberán realizar sólo en ese equipo).

Hay que recordar reiniciar el demonio *ssh* después de llevar a cabo los cambios necesarios en los ficheros de configuración indicados en cada momento, si queremos que estos sean efectivos. Para ello debemos de introducir el comando que se observa en la *ilustración 44*.

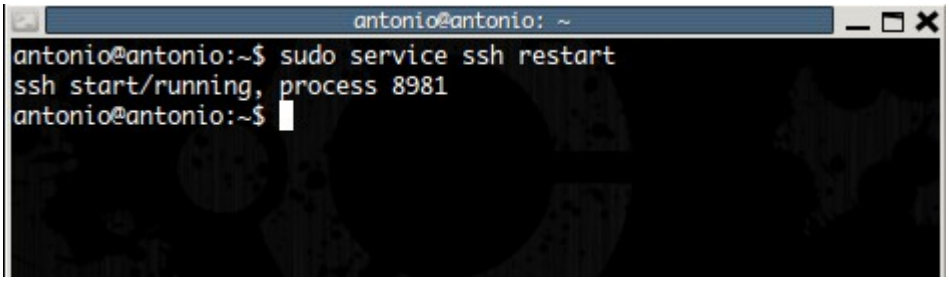

Ilustración 44: Reinicio del servidor ssh.

5.3.1.1. Autenticación mediante clave pública

Para que la autenticación sea hecha mediante el uso de certificados, debemos de crear 2 pares de claves en el cliente *ssh*, el primero de ellos, conocico como *clave privada*, será almacenado en el cliente *ssh*, mientras que el segundo, deberá ser transferido y almacenado

en el servidor *ssh* remoto. Esto se hará suponiendo que el usuario usado tanto en el cliente como en el servidor *ssh* es "*root*", y que el servidor *ssh* se encuentra en la IP 192.168.1.128.

Para generar estas claves nos valdremos del comando *ssh-keygen*, de la forma que se observa en la *ilustración 45*.

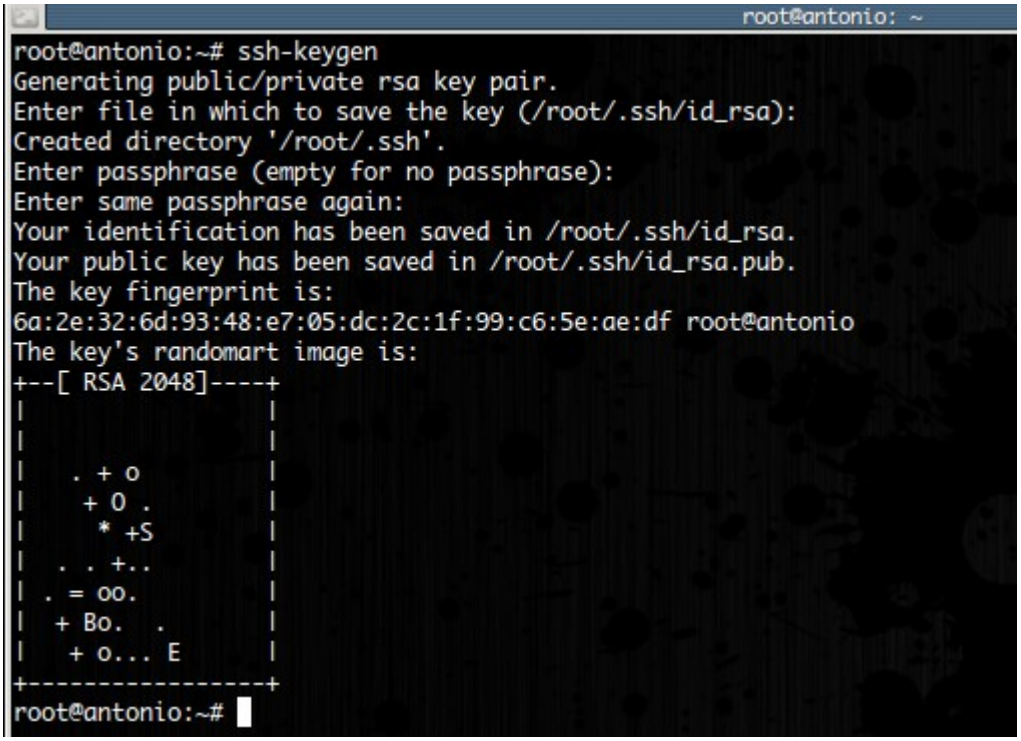

Ilustración 45: Creación de certificados.

Como podemos ver en la imagen, se pedirá un fichero donde guardar la contraseña, en nuestro caso pulsaremos *enter* para dejarlo en el archivo por defecto. A continuación se pedirá una contraseña para proteger el certificado, que debemos introducir nuevamente para cerciorarnos de su validez.

Una vez terminado el proceso, en la carpeta "root*/.ssh*" se obtendrá un nuevo archivo con la clave pública *(id_rsa.pub)* y otro con la clave privada *(id_rsa)*. A continuación, debemos de transmitir la clave pública al servidor *ssh*, con ayuda del comando *ssh-copy-id*, como se observa en la *ilustración 46*.

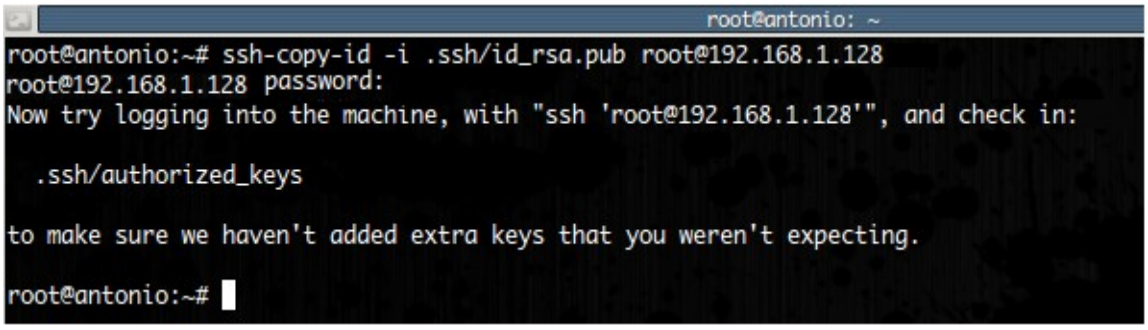

Ilustración 46: Transferencia de clave pública.

Con esto, ssh-copy-id habrá agregado la clave pública al archivo "root*/.ssh/authorized_keys*" del usuario en el servidor remoto, y la siguiente vez que nos conectemos la autenticación será automática, sin necesidad de ingresar la contraseña.

5.3.1.2. Permitiendo y denegando *host*

Para permitir y denegar los accesos, editaremos los archivos "/etc/hosts.deny" y "/etc/hosts.allow" en el servidor ssh. En el primero de ellos denegaremos todos los host y en el segundo permitiremos únicamente el cliente ssh, de forma que quedará todo denegado excepto lo que permitamos en hosts.allow.

De ésta forma, si suponemos que la IP del cliente ssh es la 192.168.1.100, procederíamos de la siguiente manera:

- Añadimos al archivo "/etc/hosts.deny " la siguiente línea:

```
ALL: ALL
```

- Añadimos al archivo "/etc/hosts.allow" la siguiente línea:

```
sshd: 192.168.1.100
```

Es imprescindible que comprendamos que los archivos host no se aplican solo al servidor ssh, sino a toda la máquina, por lo que hay que tener especial cuidado a la hora de permitir o denegar los accesos.

5.3.1.3. Cambiar el número de conexiones concurrentes no autenticadas

Ésta es una buena estrategia también, para evitar intentos de conexión. La directiva `MaxStartUps` controla el número de conexiones no autenticadas en el servidor, de ésta forma, evitamos que posibles atacantes intentasen conectarse demasiadas veces.
Editamos el archivo "/etc/ssh/sshd_config" en el servidor ssh, cambiando o añadiendo la directiva `MaxStartUps` a un valor de 1, ya que solo debería estar operando un profesor por laboratorio simultaneamente:

```
MaxStartUps 1
```

5.3.1.4. Cambiar tiempo de espera en *login*

Cuando un usuario se conecta mediante *ssh* a una máquina, dispone de 2 minutos para hacer *login*. Si durante este periodo de tiempo el usuario no es capaz de logarse exitosamente, el servidor *ssh* automáticamente se encarga de cerrar la conexión. Este tiempo de espera es, quizá, demasiado elevado, por lo que para cambiarlo se debe cambiar o añadir en el servidor *ssh* la siguiente línea en el archivo "*/etc/ssh/sshd_config*":

```
LoginGraceTime 30s
```

5.3.1.5. Desconectar cuando no haya actividad

Una vez que un usuario se ha logado exitosamente en el sistema, es aconsejable desconectarle automáticamente cuando no haya actividad durante un periodo *x* de tiempo. Para configurar esto en *OpenSSH* se debe combinar el uso de las directivas `ClientAliveCountMax` y `ClientAliveInterval` en el archivo de configuración *sshd_config*.

- `ClientAliveCountMax`. Esta directiva indica el número total de mensajes *checkalive* enviados por el servidor *ssh* sin recibir respuesta por parte del cliente *ssh*. Por defecto el valor es 3.

- `ClientAliveInternal`. Esta directiva indica el *timeout* en segundos. Después de un número de segundos *x*, el servidor *ssh* enviará un mensaje al cliente *ssh* esperando una respuesta. Por defecto el valor es 0 (el servidor *ssh* no enviará mensajes al cliente *ssh*).

Para conseguir que el cliente *ssh* se desconecte automáticamente después de 1 minuto (60 segundos), se debe modificar el fichero de configuración *"/etc/ssh/sshd_config",* cambiando o añadiendo las siguientes líneas en el servidor *ssh*:

```
ClientAliveInterval 60
ClientAliveCountMax 0
```

5.3.2. Otras prácticas de seguridad

En este apartado vamos a pasar a comentar una serie de medidas de seguridad a implementar, basándonos en el apartado 4.5.3., las cuales nos ayudarán a fortalecer la seguridad existente en el laboratorio.

5.3.2.1. Filtrado de tráfico entrante y saliente

Se debe de filtrar el flujo de datos que entra y sale de la red del laboratorio, permitiendo sólo el tráfico que se considere necesario y desechando el restante.

Para llevar a cabo esta medida, se usará *iptables* - como ya se indicó previamente -, de forma que suponiendo una estructura de red como la que se puede observar en la *ilustración 47*, en la cual hemos configurado un equipo con 2 interfaces de red (192.168.0.99 y 192.168.1.99), se usará este a modo de *firewall*.

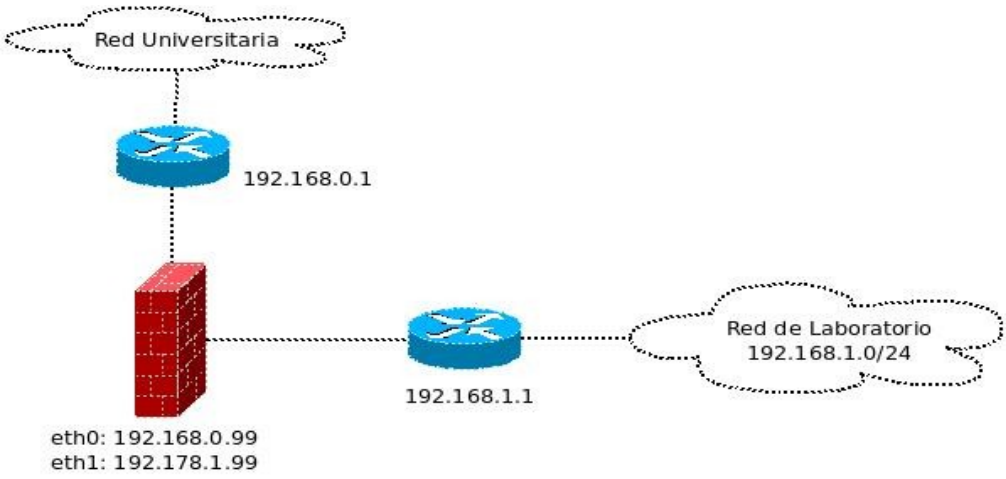

Ilustración 47: Esquema de red con Firewall.

Dado que la configuración y el uso de *iptables* se escapa del propósito del Trabajo de Fin de Carrera, sólo se enunciará mediante lenguaje natural una serie de reglas básicas, las cuales se deben escribir en el lenguaje usado por *iptables* y completar adecuadamente con otras reglas.

- Permitir ***sólo*** tráfico desde el *exterior* al *interior* con destino las máquinas usadas por los ***alumnos***, mediante el ***protocolo VNC***.

- Permitir ***sólo*** tráfico desde el *interior* al *exterior* si previamente hay una ***conexión establecida*** mediante el ***protocolo VNC***.

5.3.2.2. Monitorización de roles de los alumnos

Se debe poder controlar que los alumnos sólo ejerzan un rol simultaneamente, en caso contrario los resultados de la evaluación se podrían ver manipulados.

Para esto, se propone el uso de certificados a nivel *ssh*, que permitirán el acceso a las máquinas físicas para fortificarlas a los alumnos destinados a ello, e impidiendo esta tarea a los demás. Esta configuración se hace de la forma ya explicada en el apartado 5.3.1.1. Por lo que una vez hecha, los profesores deben distribuir los certificados correspondientes a los alumnos en función del rol que quieran que ejerza en cada momento.
Es importante que los profesores revoquen los certificados y vuelvan a crear nuevos cuando se deseen cambiar los roles de los alumnos.

5.3.2.3. Separación de las máquinas

Se recomienda usar máquinas físicas diferentes a la hora de desplegar el escenario, con respecto a las usadas por los alumnos para llevar a cabo las prácticas. Con esto ganamos un nivel más de seguridad, al no tener los alumnos acceso físico a las máquinas sobre las que

está montado el escenario.

Para llevar a cabo esta medida, se debe reservar una serie de ordenadores dentro del laboratorio para el despliegue de los escenarios planteados por los profesores, y otra serie de ordenadores para que los alumnos lleven a cabo las tareas que se les indique.

5.3.2.4. *Software* actualizado

Ya que se va a usar *software* muy específico, se recomienda estar al tanto de actualizaciones de seguridad que puedan surgir, para así evitar posibles brechas se seguridad.

Para llevar a cabo esta medida, los profesores deben de revisar las actualizaciones disponibles, y aplicarlas en cada máquina del laboratorio. A continuación se muestra en las *ilustraciones 48 y 49* los comandos que llevan a cabo estas funcionalidades.

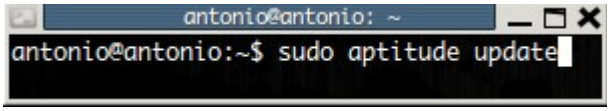

Ilustración 48: Búsqueda de actualizaciones.

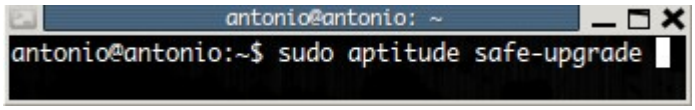

Ilustración 49: Instalación de actualizaciones.

Estos comandos deben ser ejecutados en todas las máquinas del laboratorio, verificando previamente que se disponga de acceso a *internet*.

6

PRUEBAS

En este capítulo se van a presentar una serie de pruebas para verificar que la instalación y configuración de los capítulos anteriores funcionan de la manera esperada.

Para llevar a cabo esta tarea tenemos 2 opciones:

- Diseñar varios escenarios pequeños, cada uno con ciertos elementos de entre todos los posibles. De forma que se acabarían usando todos los elementos en uno u otro escenario.

- Diseñar un solo escenario con todos los elementos posibles. De forma que aunque sea más laborioso definirlo, tendremos en un único escenario todos los elementos y se podrá verificar que interaccionan correctamente entre ellos.

Se ha elegido la segunda aproximación ya que esta sería la forma de trabajar más acorde con la realidad en un laboratorio, donde los profesores desplegarían un sólo escenario complejo, que contendría todos los elementos posibles para simular una red empresarial existente en la realidad.

En un primer apartado se harán las pruebas de *definición y parseo* del escenario, en las que se incluirán casos de error para ver el correcto manejo de estos por la aplicación desarrollada.

A continuación verificaremos el correcto funcionamiento de la fase de *despliegue* y la posterior *conectividad* entre las máquinas virtuales que compongan el escenario.

En un tercer apartado, se realizarán las pruebas necesarias para verificar que los *servicios* y los *eventos* creados son lanzados correctamente y en el periodo de tiempo indicado en la definición del escenario.

Finalemente, se probará el modo en que los profesores se pueden conectar a las máquinas para realizar la gestión del escenario.

6.1. Definición y parseo

En este apartado se van a llevar a cabo las pruebas necesarias para cerciorarnos de que la aplicación desarrollada para este Trabajo de Fin de Carrera valida correctamente los archivos de configuración que necesita. Para esto, vamos a relizar una división de las pruebas, como se indica a continuación.

- En un primer apartado haremos las pruebas necesarias sobre el archivo "*config.cfg*", probando en primer lugar a definirlo de una manera incorrecta, y posteriormente de una manera correcta.

- A continuación, nos centraremos en las pruebas del archivo "*templates.xml*", como en el caso anterior, en primer lugar se definirá de manera incorrecta, y posteriormente de una manera correcta.

- En el siguiente apartado, se harán las pruebas del script "*vnc_send_keys.py*". De igual manera, se probará a introducirle comandos incorrectos y después comandos correctos.

- El cuarto apartado estará dedicado a la probatura de la validación de argumentos por el *programa desarrollado* para este Trabajo de Fin de Carrera, probando a introducirle argumentos incorrectos y posteriormente correctos.

- Finalmente, se dedicará un apartado a la prueba del *escenario* creado, donde se definirá en primer lugar un escenario no válido, y a continuación uno válido que será el usado en el resto del capítulo para las pruebas.

6.1.1. Archivo "*config.cfg*"

Vamos a verificar el tratamiento por parte de nuestra aplicación de este archivo, definiendo casos de prueba incorrectos en un principio, y posteriormente un caso de prueba correcto.

Nombre	Archivo configuración inválido (sin cabeceras)
ID	CP 1
Descripción	Se define un archivo de configuración sin ninguna cabecera, obteniendo el mensaje de error que se observa en la *ilustración 50*.

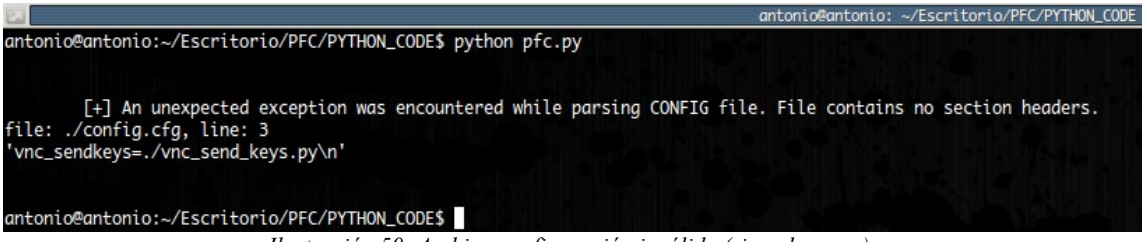

Ilustración 50: Archivo configuración inválido (sin cabeceras).

Nombre	Archivo configuración inválido (cabecera no *main*)
ID	CP 2
Descripción	Se define un archivo de configuración con una cabecera que no es *main*, obteniendo el mensaje de error que se observa en la *ilustración 51*.

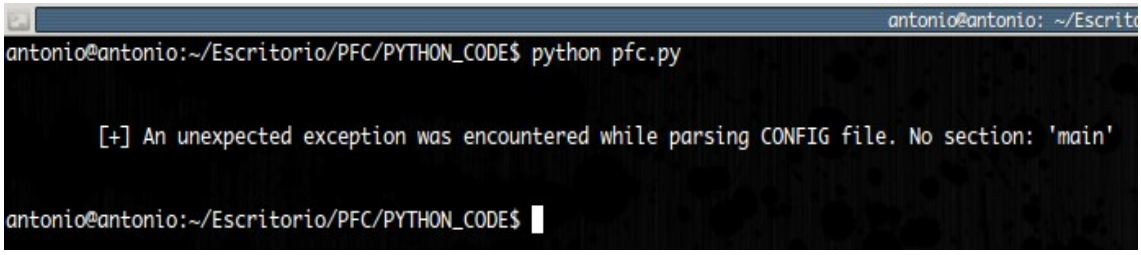

Ilustración 51: Archivo configuración inválido (cabecera no main).

Nombre	Archivo configuración inválido (sin el atributo *vnc_sendkeys*)
ID	CP 3
Descripción	Se define un archivo de configuración sin el atributo *vnc_sendkeys,* obteniendo el mensaje de error que se observa en la *ilustración 52*.

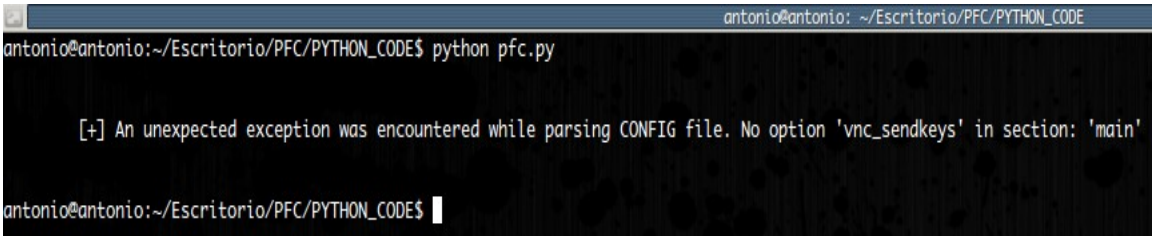

Ilustración 52:Archivo configuración inválido (sin el atributo vnc_sendkeys).

Nombre	Archivo configuración inválido (sin el atributo *templates*)
ID	CP 4
Descripción	Se define un archivo de configuración sin el atributo *templates,* obteniendo el mensaje de error que se observa en la *ilustración 53*.

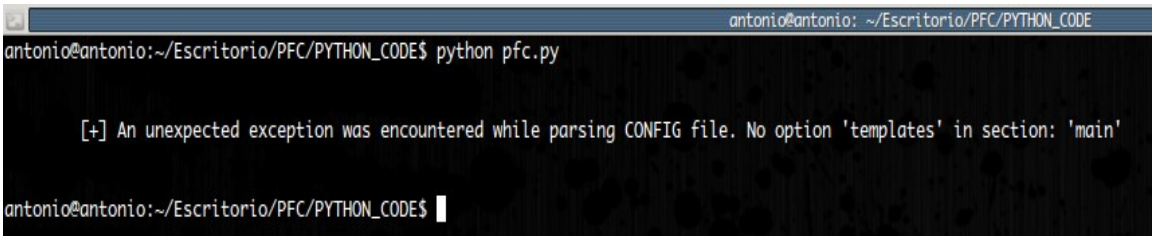

Ilustración 53: Archivo configuración inválido (sin el atributo templates).

Nombre	Archivo configuración inválido (sin el atributo *input_stage*)
ID	CP 5
Descripción	Se define un archivo de configuración sin el atributo *input_stage,* obteniendo el mensaje de error que se observa en la *ilustración 54*.

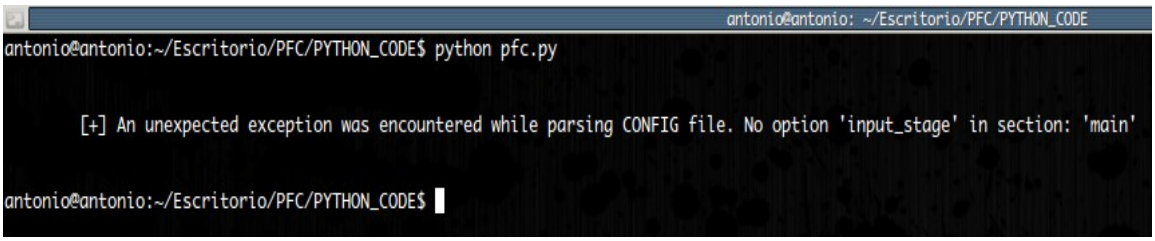

Ilustración 54: Archivo configuración inválido (sin el atributo input_stage).

Nombre	Archivo configuración inválido (sin el atributo *output_script*)
ID	CP 6
Descripción	Se define un archivo de configuración sin el atributo *output_script,* obteniendo el mensaje de error que se observa en la *ilustración 55.*

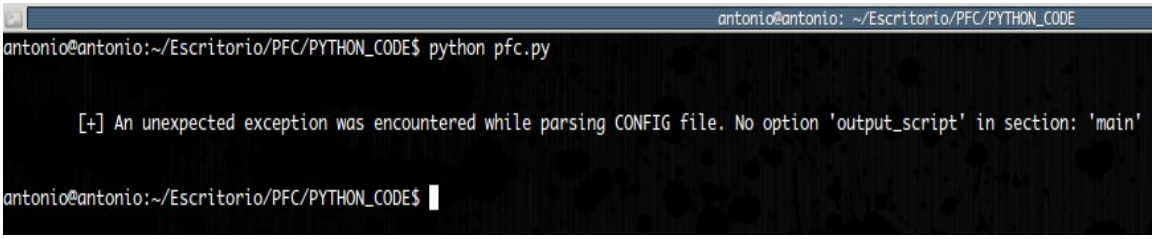

Ilustración 55: Archivo configuración inválido (sin el atributo output_script).

Nombre	Archivo configuración válido
ID	CP 7
Descripción	Se define un archivo de configuración válido, como el indicado en la *ilustración 22*. En este caso la aplicación no emite ningún mensaje de error.

6.1.2. Archivo "*templates.xml*"

Vamos a verificar el tratamiento por parte de nuestra aplicación de este archivo, definiendo casos de prueba incorrectos en un principio, y posteriormente un caso de prueba correcto.

Nombre	Archivo de plantillas inválido (sin el *tag* raiz *templates*)
ID	CP 8
Descripción	Se define un archivo de plantillas sin el *tag* raiz *templates*, obteniendo el mensaje de error que se observa en la *ilustración 56*.

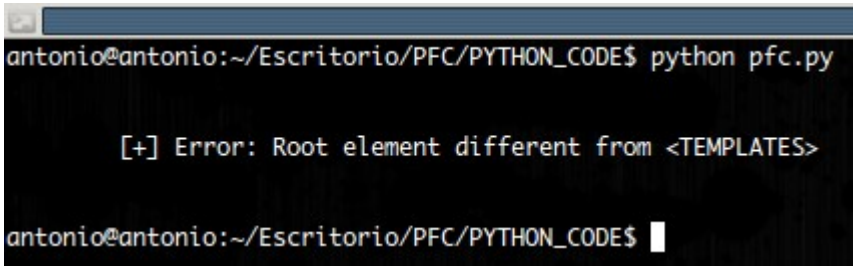

Ilustración 56: Archivo de plantillas inválido (sin el tag raiz templates).

Nombre	Archivo de plantillas inválido (sin el *tag vm*)
ID	CP 9
Descripción	Se define un archivo de plantillas sin el *tag vm*, obteniendo el mensaje de error que se observa en la *ilustración 57*.

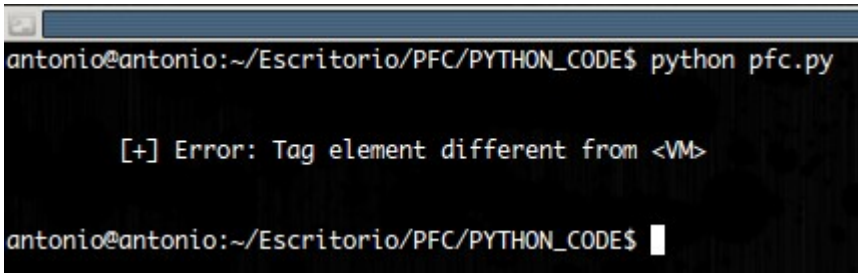

Ilustración 57: Archivo de plantillas inválido (sin el tag vm).

Nombre	Archivo de plantillas inválido (sin el *tag service* o el *tag attack*)
ID	CP 10
Descripción	Se define un archivo de plantillas sin el *tag service* o el *tag attack*, obteniendo el mensaje de error que se observa en la *ilustración 58*.

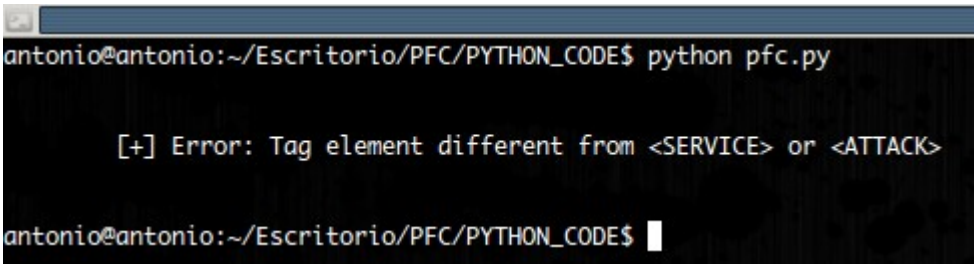

Ilustración 58: Archivo de plantillas inválido (sin el tag service o el tag attack).

Nombre	Archivo de plantillas inválido (sin el *tag param*)
ID	CP 11
Descripción	Se define un archivo de plantillas sin el *tag param* dentro del *tag service*, obteniendo el mensaje de error que se observa en la *ilustración 59*.

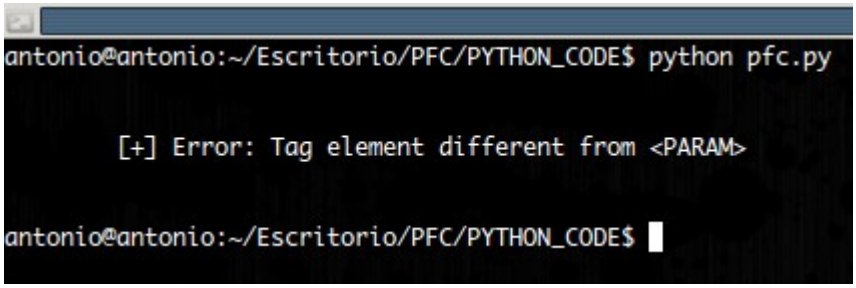

Ilustración 59: Archivo de plantillas inválido (sin el tag param).

Nombre	Archivo de plantillas inválido (sin el *tag type*)
ID	CP 12
Descripción	Se define un archivo de plantillas sin el *tag type* dentro del *tag attack*, obteniendo el mensaje de error que se observa en la *ilustración 60*.

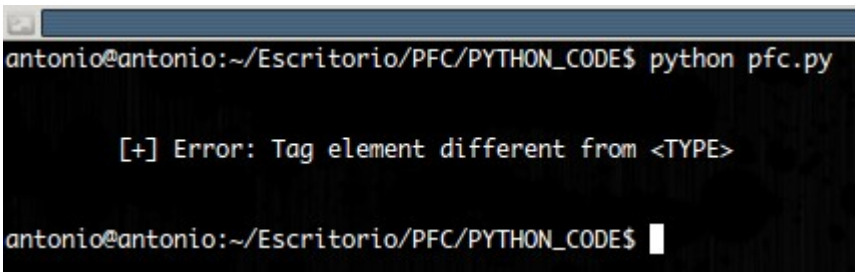

Ilustración 60: Archivo de plantillas inválido (sin el tag type).

Nombre	Archivo de plantillas inválido (sin el *tag param*)
ID	CP 13
Descripción	Se define un archivo de plantillas sin el *tag param* dentro del *tag attack*, obteniendo el mensaje de error que se observa en la *ilustración 61*.

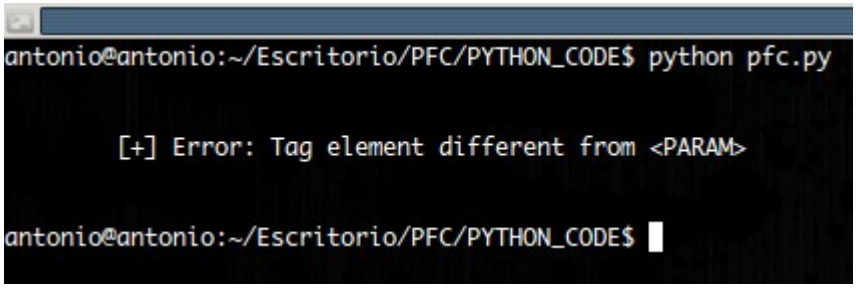

Ilustración 61: Archivo de plantillas inválido (sin el tag param).

Nombre	Archivo de plantillas inválido (sin los tags necesarios en el elemento vm)
ID	CP 14
Descripción	Se define un archivo de plantillas sin los tags necesarios en el elemento vm, obteniendo el mensaje de error que se observa en la *ilustración 62*.

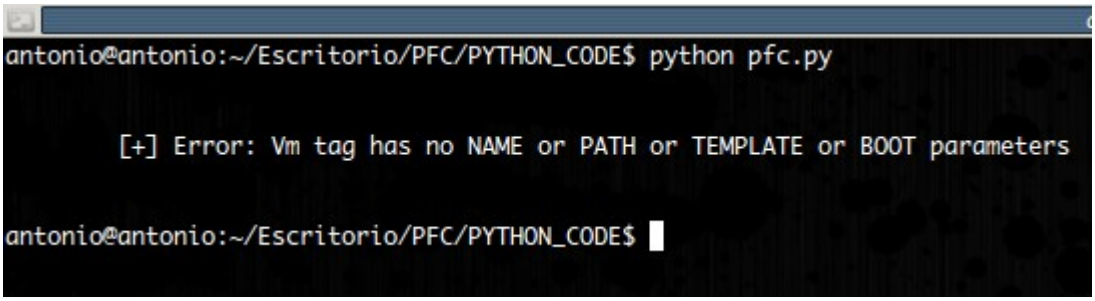

Ilustración 62: Archivo de plantillas inválido (sin los tags necesarios en el elemento vm).

Nombre	Archivo de plantillas inválido (sin los *tags* necesarios en el elemento *service*)
ID	CP 15
Descripción	Se define un archivo de plantillas sin los *tags* necesarios en el elemento *service*, obteniendo el mensaje de error que se observa en la *ilustración 63*.

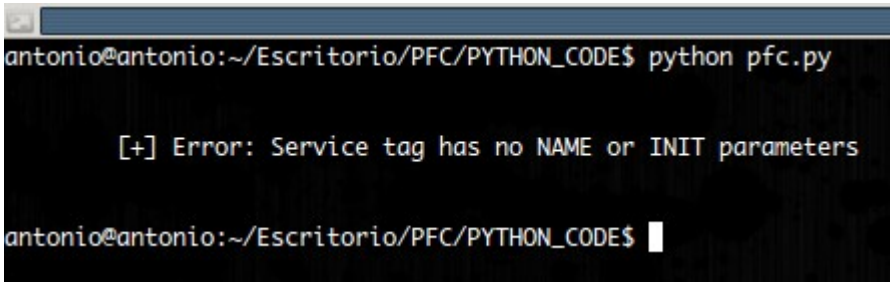

Ilustración 63: Archivo de plantillas inválido (sin los tags necesarios en el elemento service).

Nombre	Archivo de plantillas inválido (sin los *tags* necesarios en el elemento *param*)
ID	CP 16
Descripción	Se define un archivo de plantillas sin los *tags* necesarios en el elemento *param* del *tag service*, obteniendo el mensaje de error que se observa en la *ilustración 64*.

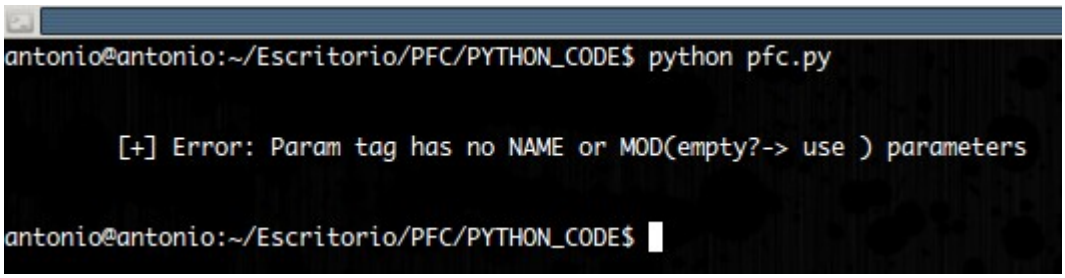

Ilustración 64: Archivo de plantillas inválido (sin los tags necesarios en el elemento param).

Nombre	Archivo de plantillas inválido (sin los *tags* necesarios en el elemento *attack*)
ID	CP 17
Descripción	Se define un archivo de plantillas sin los *tags* necesarios en el elemento *attack*, obteniendo el mensaje de error que se observa en la *ilustración 65*.

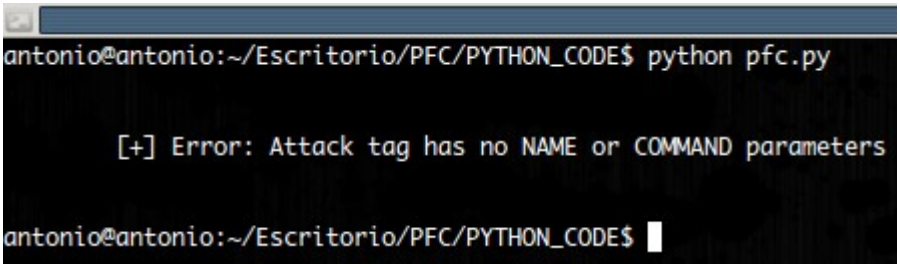

Ilustración 65: Archivo de plantillas inválido (sin los tags necesarios en el elemento attack).

Nombre	Archivo de plantillas inválido (sin los *tags* necesarios en el elemento *param*)
ID	CP 18
Descripción	Se define un archivo de plantillas sin los *tags* necesarios en el elemento *param* del *tag attack*, obteniendo el mensaje de error que se observa en la ilustración 66.

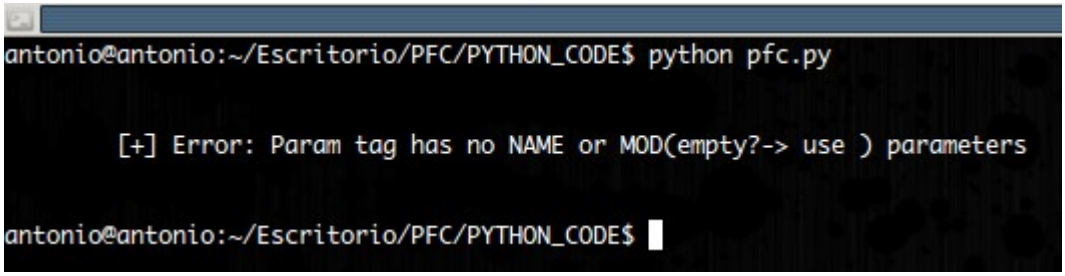

Ilustración 66: Archivo de plantillas inválido (sin los tags necesarios en el elemento param).

Nombre	Archivo de plantillas inválido (sin los *tags* necesarios en el elemento *type*)
ID	CP 19
Descripción	Se define un archivo de plantillas sin los *tags* necesarios en el elemento *type*, obteniendo el mensaje de error que se observa en la *ilustración 67*.

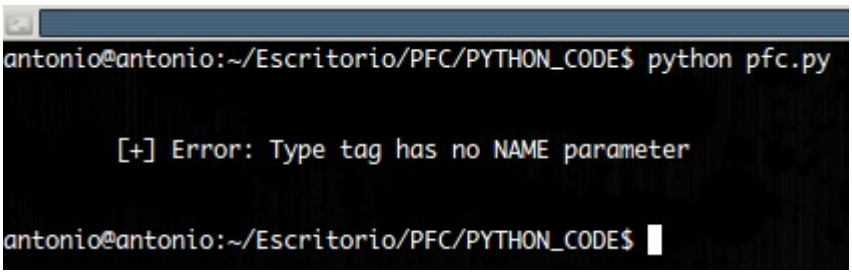

Ilustración 67: Archivo de plantillas inválido (sin los tags necesarios en el elemento type).

Nombre	Archivo de plantillas válido
ID	CP 20
Descripción	Se define un archivo de plantillas válido, como el indicado en la *ilustración 20*. En este caso la aplicación no emite ningún mensaje de error.

6.1.3. *Script "vnc_send_keys"*

Vamos a verificar el tratamiento por parte de nuestra aplicación de este *script*, definiendo casos de prueba incorrectos en un principio, y posteriormente un caso de prueba correcto.

Nombre	Archivo de plantillas inválido (parámetros incorrectos)
ID	CP 21
Descripción	Se ejecuta el *script* sin ningún parámetro o con un número de parámetros incorrecto, obteniendo el mensaje de error que se observa en la *ilustración 68*.

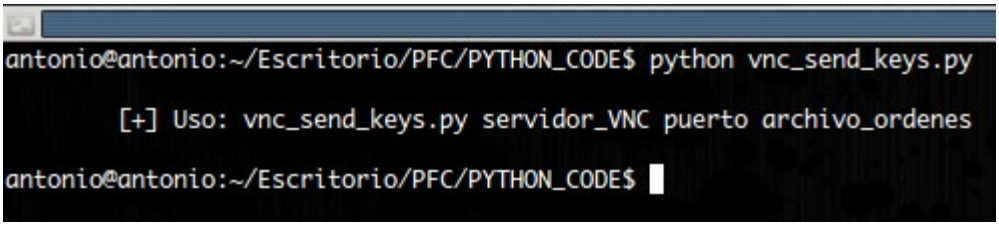

Ilustración 68: Archivo de plantillas inválido (parámetros incorrectos).

Nombre	Archivo de plantillas inválido *(host* incorrecto*)*
ID	CP 22
Descripción	Se ejecuta el *script* intentando conectar a un *host* inexistente o en el que no hay ningún servidor *VNC* escuchando, obteniendo el mensaje de error que se observa en la *ilustración 69*.

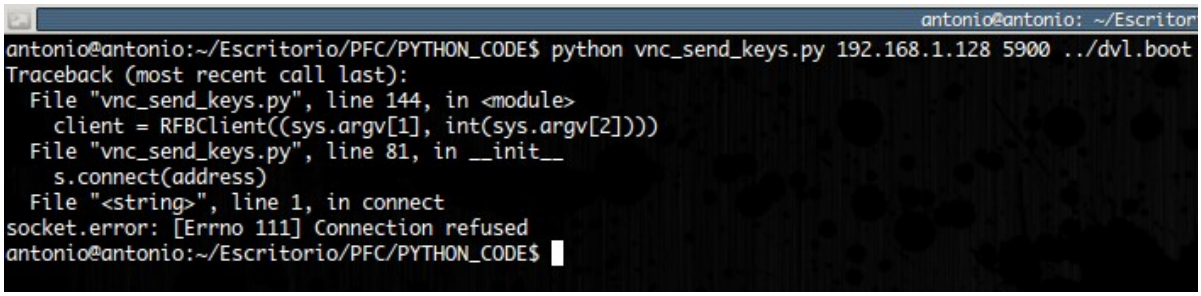

Ilustración 69: Archivo de plantillas inválido (host incorrecto).

Nombre	Archivo de plantillas inválido (comando no válido)
ID	CP 23
Descripción	Se ejecuta el *script* mandando un comando no válido, obteniendo el mensaje de error que se observa en la *ilustración 70*.

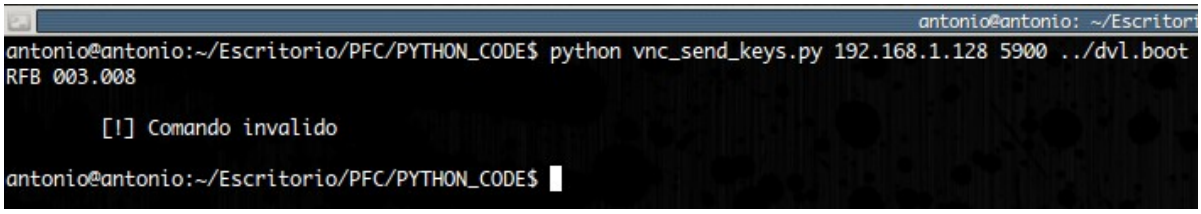

Ilustración 70: Archivo de plantillas inválido (comando no válido).

Nombre	Archivo de plantillas válido
ID	CP 24
Descripción	Se ejecuta el *script* con comandos válidos, como los indicados en la *ilustración 24*. En este caso la aplicación no emite ningún mensaje de error.

6.1.4. Aplicación desarrollada

Vamos a verificar el tratamiento por parte de nuestra aplicación de los parámetros pasados, definiendo casos de prueba incorrectos en un principio, y posteriormente un caso de prueba correcto.

Nombre	Parámetros inválidos en aplicación desarrollada
ID	CP 25
Descripción	Se ejecuta la aplicación pasándole parámetros no válidos, obteniendo el mensaje de error que se observa en la *ilustración 71*.

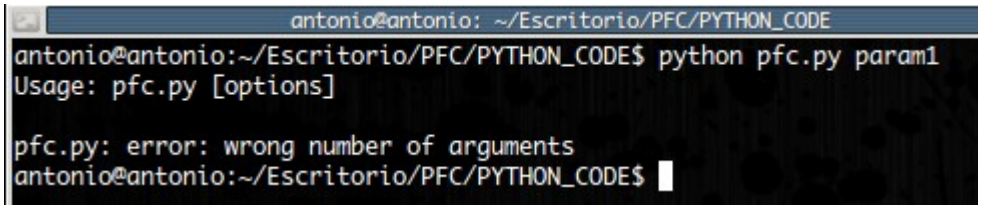

Ilustración 71: Parámetros inválidos en aplicación desarrollada.

Nombre	Parámetro con valor inválido en aplicación desarrollada (no existente)
ID	CP 26
Descripción	Se ejecuta la aplicación pasándole un parámetro con valor no existente, obteniendo el mensaje de error que se observa en la *ilustración 72*.

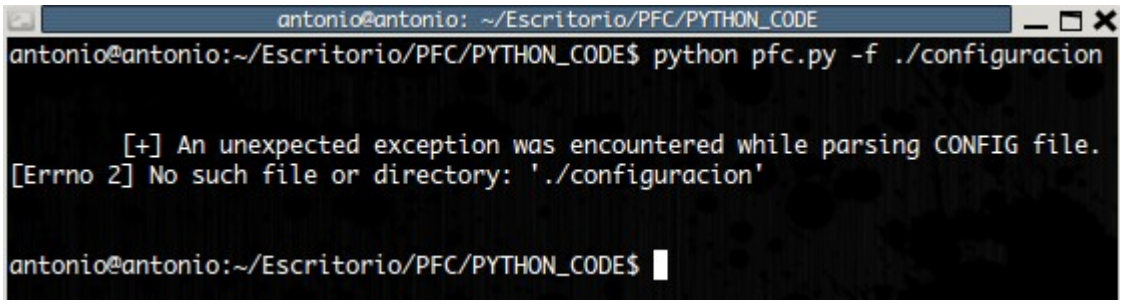

Ilustración 72: Parámetro con valor inválido en aplicación desarrollada (no existente).

Nombre	Parámetro con valor inválido en aplicación desarrollada (archivo por defecto no existente)
ID	CP 27
Descripción	Se ejecuta la aplicación con el valor del parámetro por defecto, el cual no existente, obteniendo el mensaje de error que se observa en la *ilustración 73*.

Ilustración 73: Parámetro con valor inválido en aplicación desarrollada (archivo por defecto no existente).

Nombre	Parámetro con valor válido en aplicación desarrollada
ID	CP 28
Descripción	Se ejecuta la aplicación con un parámetro válido correspondiente a la ayuda, obteniendo el mensaje informativo que se observa en la *ilustración 74*.

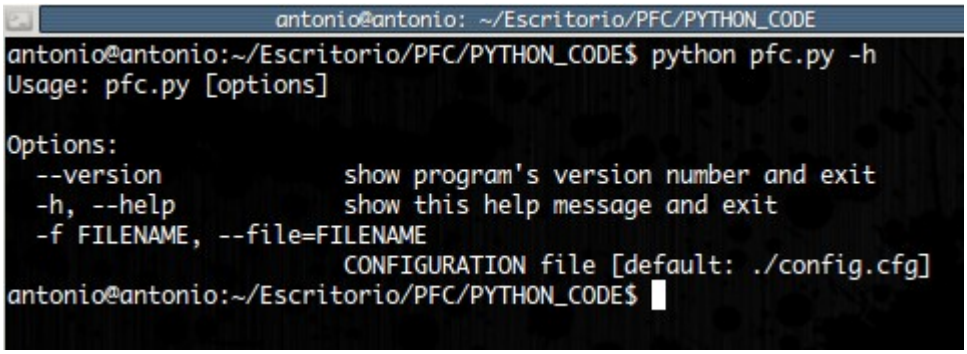

Ilustración 74: Parámetro con valor válido en aplicación desarrollada.

Nombre	Parámetro con valor válido en aplicación desarrollada
ID	CP 29
Descripción	Se ejecuta la aplicación con un parámetro válido, sin obtenerse ningún mensaje de error y continuando la ejecución normal del programa

6.1.5. Archivo "*stage.xml*"

Vamos a verificar el tratamiento por parte de nuestra aplicación del archivo donde se desfine el escenario a implementar, definiendo casos de prueba incorrectos en un principio, y posteriormente un caso de prueba correcto.

Nombre	Archivo de escenario inválido (sin el *tag* raiz *stage*)
ID	CP 30
Descripción	Se define un archivo de escenario sin el *tag* raiz *stage*, obteniendo el mensaje de error que se observa en la *ilustración 75*.

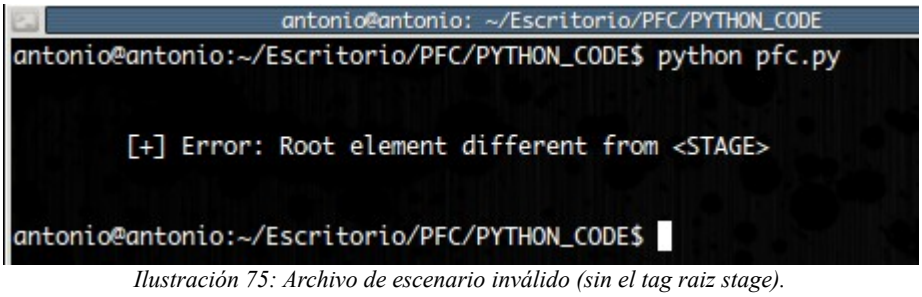

Ilustración 75: Archivo de escenario inválido (sin el tag raiz stage).

Nombre	Archivo de escenario inválido (sin el *tag segment*)
ID	CP 31
Descripción	Se define un archivo de escenario sin el *tag segment*, obteniendo el mensaje de error que se observa en la *ilustración 76*.

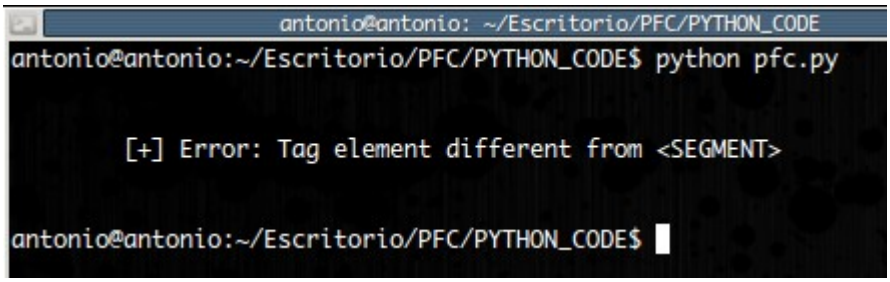

Ilustración 76: Archivo de escenario inválido (sin el tag segment).

Nombre	Archivo de escenario inválido (sin el *tag host* o *gateway*)
ID	CP 32
Descripción	Se define un archivo de escenario sin el *tag host* o *gateway* dentro de un segmento, obteniendo el mensaje de error que se observa en la *ilustración 77*.

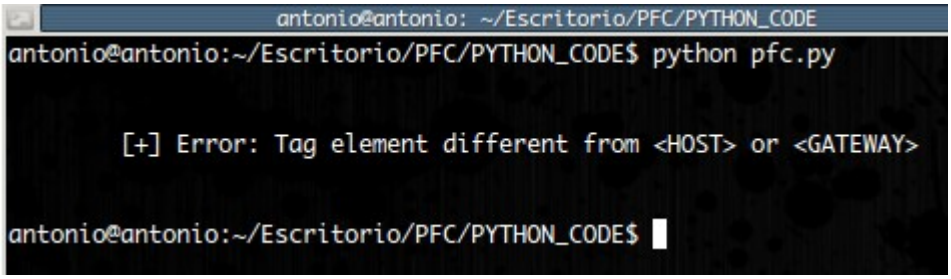

Ilustración 77: Archivo de escenario inválido (sin el tag host o gateway).

Nombre	Archivo de escenario inválido (sin el *tag route*, *service* o *event*)
ID	CP 33
Descripción	Se define un archivo de escenario sin el *tag route*, *service* o *event* dentro de un *host*, obteniendo el mensaje de error que se observa en la *ilustración 78*.

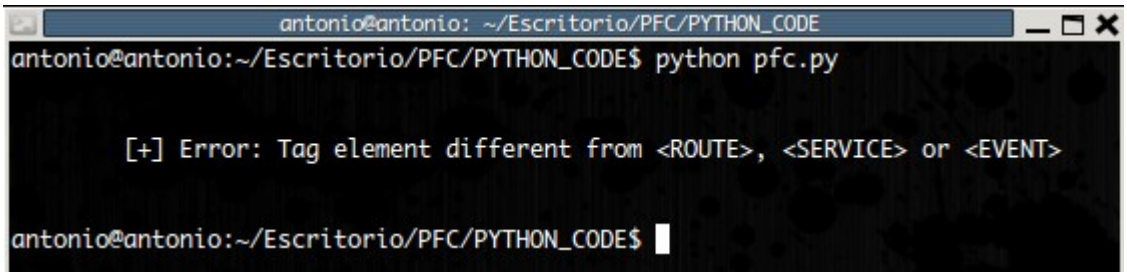

Ilustración 78: Archivo de escenario inválido (sin el tag route, service o event).

Nombre	Archivo de escenario inválido (sin el *tag attack*)
ID	CP 34
Descripción	Se define un archivo de escenario sin el *tag attack* dentro de un evento, obteniendo el mensaje de error que se observa en la *ilustración 79*.

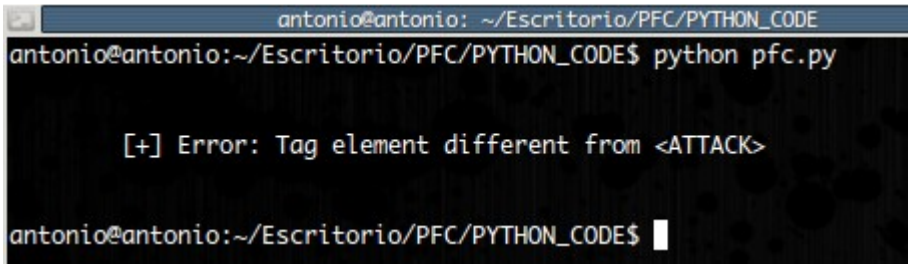

Ilustración 79: Archivo de escenario inválido (sin el tag attack).

Nombre	Archivo de escenario inválido (sin el *tag route* o *ipchains*)
ID	CP 35
Descripción	Se define un archivo de escenario sin el *tag route* o *ipchains* dentro de un *gateway*, obteniendo el mensaje de error que se observa en la *ilustración 80*.

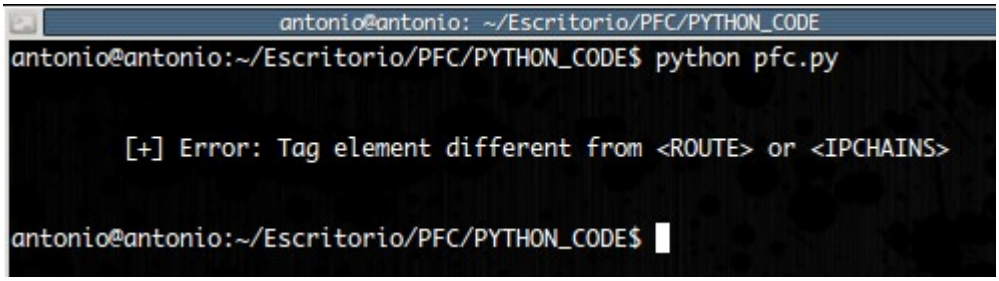

Ilustración 80: Archivo de escenario inválido (sin el tag route o ipchains).

Nombre	Archivo de escenario inválido (sin un atributo necesario en el *tag segment*)
ID	CP 36
Descripción	Se define un archivo de escenario sin un atributo necesario en el *tag segment*, obteniendo el mensaje de error que se observa en la *ilustración 81*.

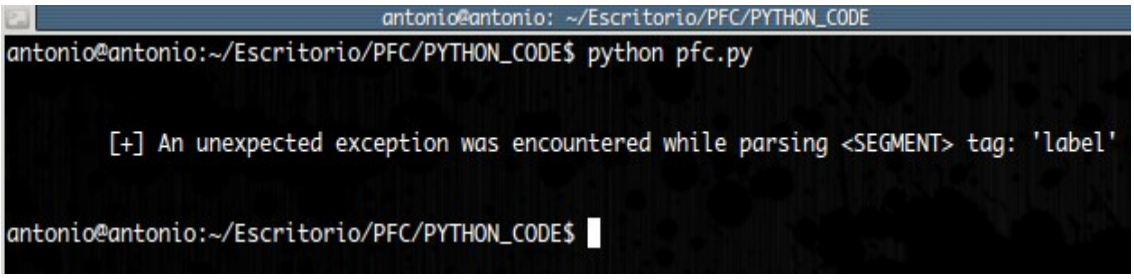

Ilustración 81: Archivo de escenario inválido (sin un atributo necesario en el tag segment).

Nombre	Archivo de escenario inválido (sin un atributo necesario en el *tag host*)
ID	CP 37
Descripción	Se define un archivo de escenario sin un atributo necesario en el *tag host*, obteniendo el mensaje de error que se observa en la *ilustración 82*.

Ilustración 82: Archivo de escenario inválido (sin un atributo necesario en el tag host).

Nombre	Archivo de escenario inválido (sin un atributo necesario en el *tag route*)
ID	CP 38
Descripción	Se define un archivo de escenario sin un atributo necesario en el *tag route*, obteniendo el mensaje de error que se observa en la *ilustración 83*.

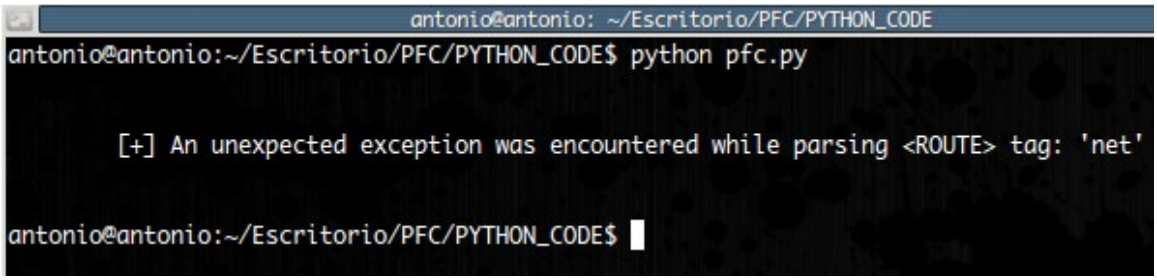

Ilustración 83: Archivo de escenario inválido (sin un atributo necesario en el tag route).

Nombre	Archivo de escenario inválido (sin un atributo necesario en el *tag host*)
ID	CP 39
Descripción	Se define un archivo de escenario sin un atributo necesario en el *tag host*, obteniendo el mensaje de error que se observa en la *ilustración 84*.

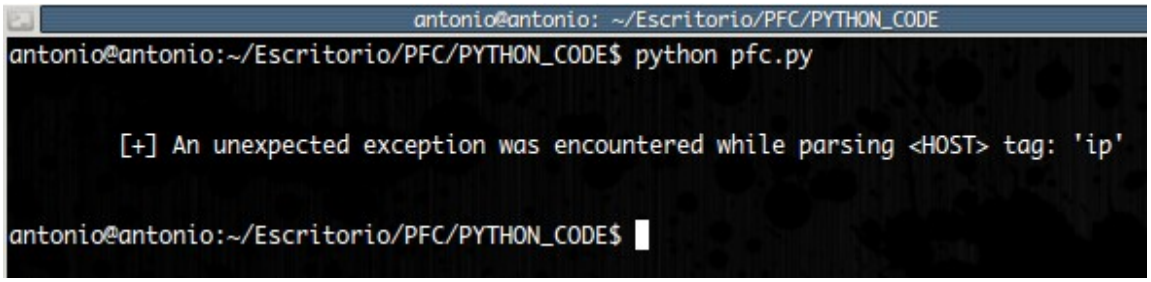

Ilustración 84: Archivo de escenario inválido (sin un atributo necesario en el tag host).

Nombre	Archivo de escenario inválido (sin un atributo necesario en el *tag service*)
ID	CP 40
Descripción	Se define un archivo de escenario sin un atributo necesario en el *tag service*, obteniendo el mensaje de error que se observa en la *ilustración 85*.

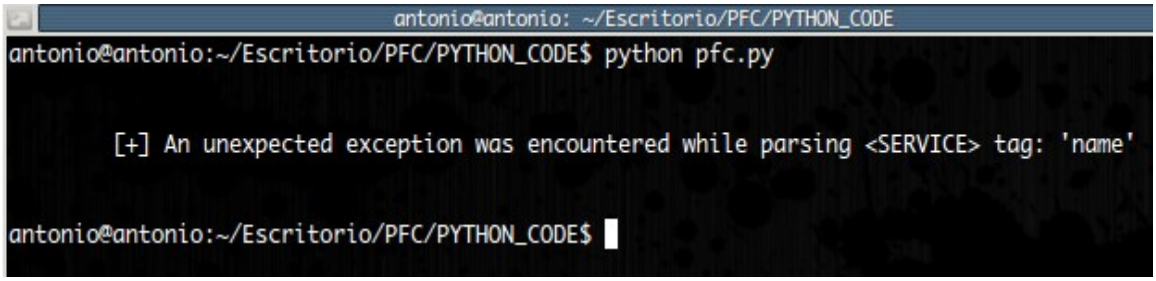

Ilustración 85: Archivo de escenario inválido (sin un atributo necesario en el tag service).

Nombre	Archivo de escenario inválido (sin un atributo necesario en el *tag event*)
ID	CP 41
Descripción	Se define un archivo de escenario sin un atributo necesario en el *tag event*, obteniendo el mensaje de error que se observa en la *ilustración 86*.

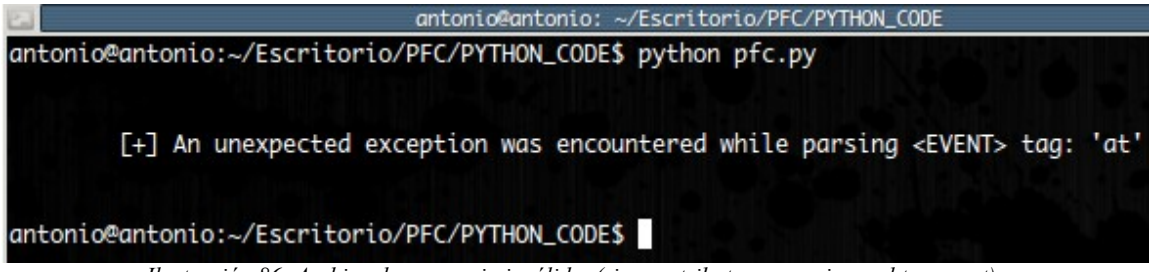

Ilustración 86: Archivo de escenario inválido (sin un atributo necesario en el tag event).

Nombre	Archivo de escenario inválido (sin un atributo necesario en el *tag attack*)
ID	CP 42
Descripción	Se define un archivo de escenario sin un atributo necesario en el *tag attack*, obteniendo el mensaje de error que se observa en la *ilustración 87*.

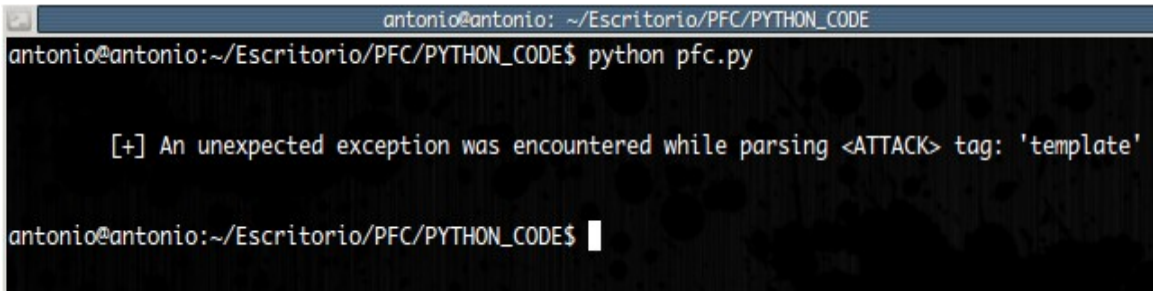

Ilustración 87: Archivo de escenario inválido (sin un atributo necesario en el tag attack).

Nombre	Archivo de escenario inválido (sin un atributo necesario en el *tag gateway*)
ID	CP 43
Descripción	Se define un archivo de escenario sin un atributo necesario en el *tag gateway*, obteniendo el mensaje de error que se observa en la *ilustración 88*.

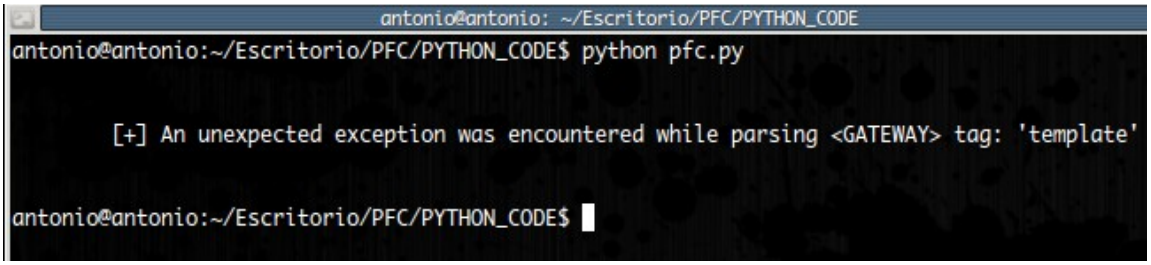

Ilustración 88: Archivo de escenario inválido (sin un atributo necesario en el tag gateway).

Nombre	Archivo de escenario inválido (sin un atributo necesario en el *tag ipchains*)
ID	CP 44
Descripción	Se define un archivo de escenario sin un atributo necesario en el *tag ipchains*, obteniendo el mensaje de error que se observa en la *ilustración 89*.

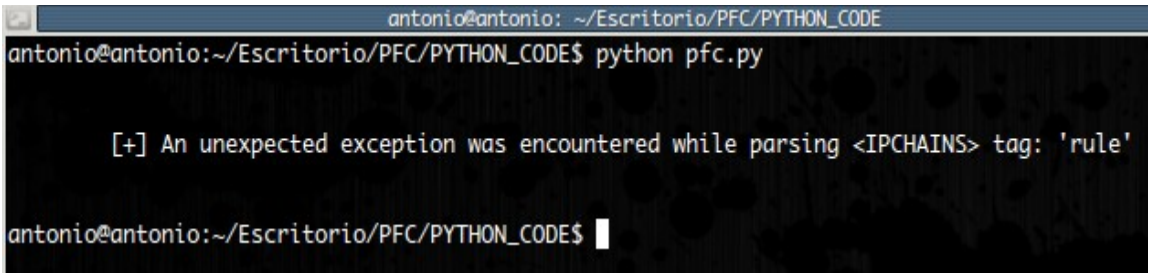

Ilustración 89: Archivo de escenario inválido (sin un atributo necesario en el tag ipchains).

Nombre	Archivo de escenario válido
ID	CP 44
Descripción	Se define un archivo de escenario válido, como el que se definió en la figura 4.4., sin obtenerse ningún mensaje de error y continuando la aplicación su ejecución.

Antes de continuar, se ha representado gráficamente el escenario definido tal y como se indicó en el caso de prueba 44. Este escenario, que será el usado para el resto de las pruebas en este capítulo, se puede observar en la *ilustración 90*.

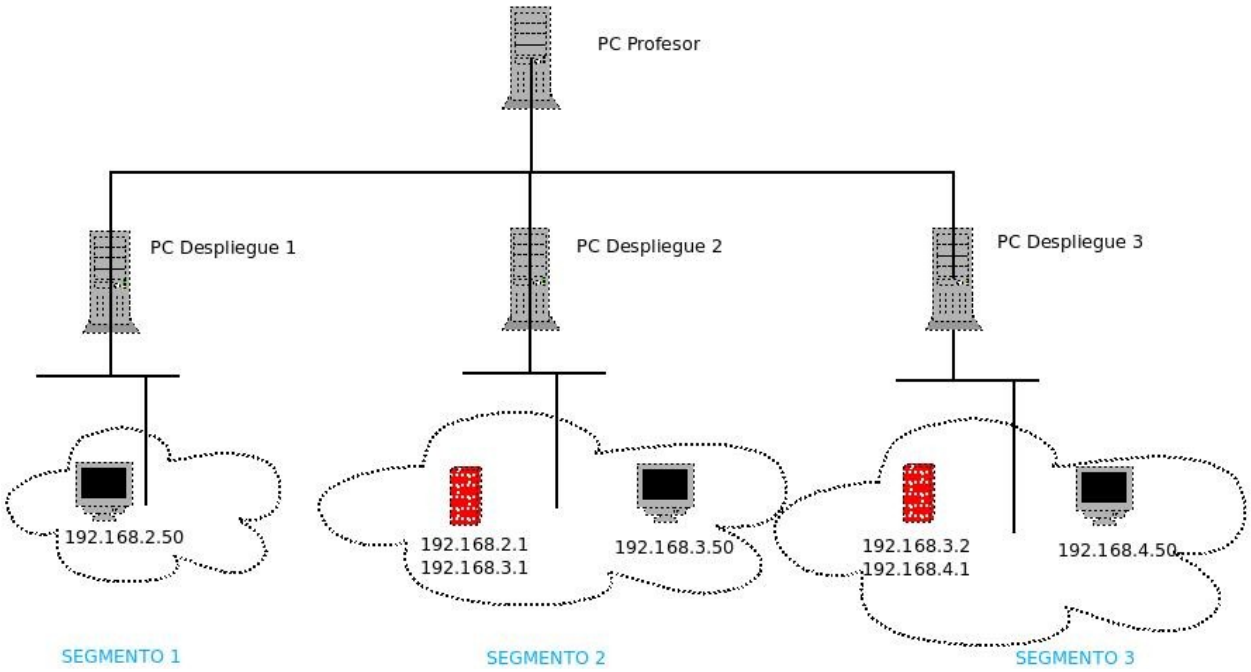

Ilustración 90: Escenario definido.

6.2. Despliegue y conectividad

En este apartado se van a realizar las pruebas relacionadas con el despliegue y la conectividad del escenario represantado en la figura 6.45.

En primer lugar realizaremos un seguimiento detallado del *despliegue*, el cual se basa en el *script* de salida generado por nuestra aplicación.

A continuación realizaremos una serie de pruebas para verificar que el escenario desplegado en el laboratorio ha sido configurado correctamente, de forma que los diferentes elementos que lo conforman poseen *conectividad* entre ellos.

6.2.1. Despliegue

En esta fase, el profesor ya habrá generado el *script* de salida a partir del escenario definido, por lo que lo único que deberá hacer es ejecutarlo y esperar a que se vaya desplegando, a continuación vamos a ir haciendo un seguimiento gráfico de las diferentes etapas comentándolas brevemente.

Nombre	Creación de las máquinas virtuales
ID	CP 45
Descripción	En primer lugar, el profesor ejecutará el *script* y automáticamente se irán generando las diferentes máquinas virtuales a lo largo de la red de laboratorio, en nuestro caso 5 máquinas virtuales. Esto se puede observar en la *ilustración 91*, donde se ve la creación de todas las máquinas virtuales, y como efectivamente han sido levantadas según aparece en el programa de gestión *virt-manager* a la derecha de la figura.

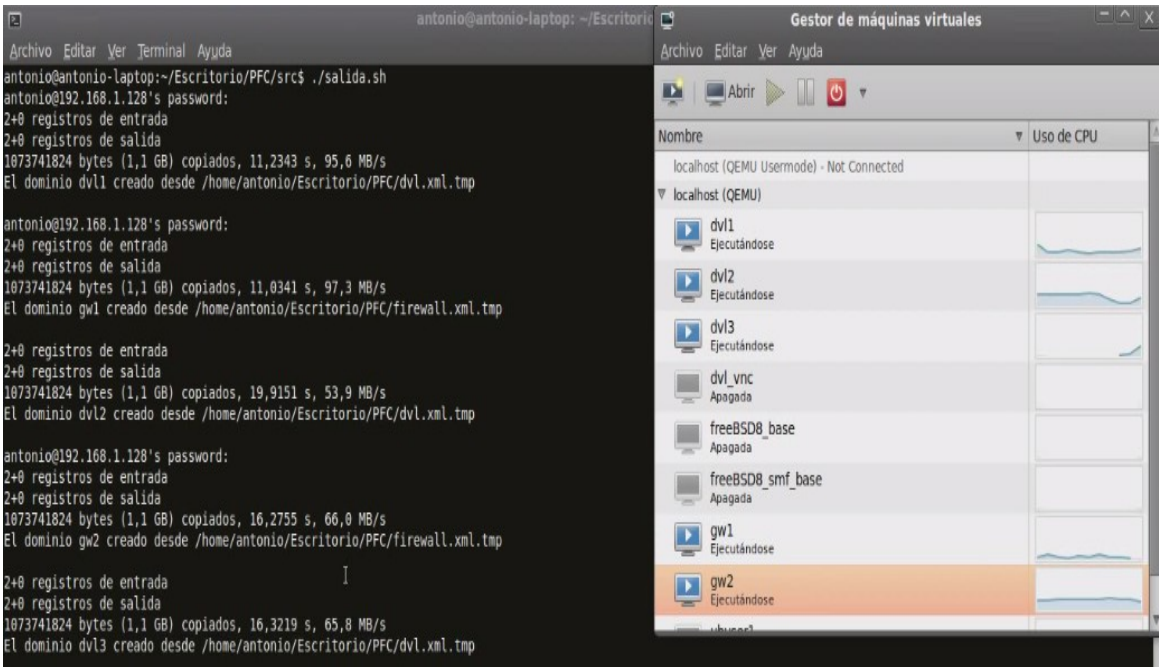

Ilustración 91: Creación de las máquinas virtuales.

Nombre	Creación de las máquinas virtuales
ID	CP 46
Descripción	Una vez que se han levantado las máquinas, se ejecutará la secuencia de comandos de arranque de cada máquina virtual a través del *script* creado para ello. Podemos ver un ejemplo de esto en la *ilustración 92*, en donde automáticamente se ha introducido el usuario y contraseña del sistema operativo, y se ha configurado el interfaz de red.

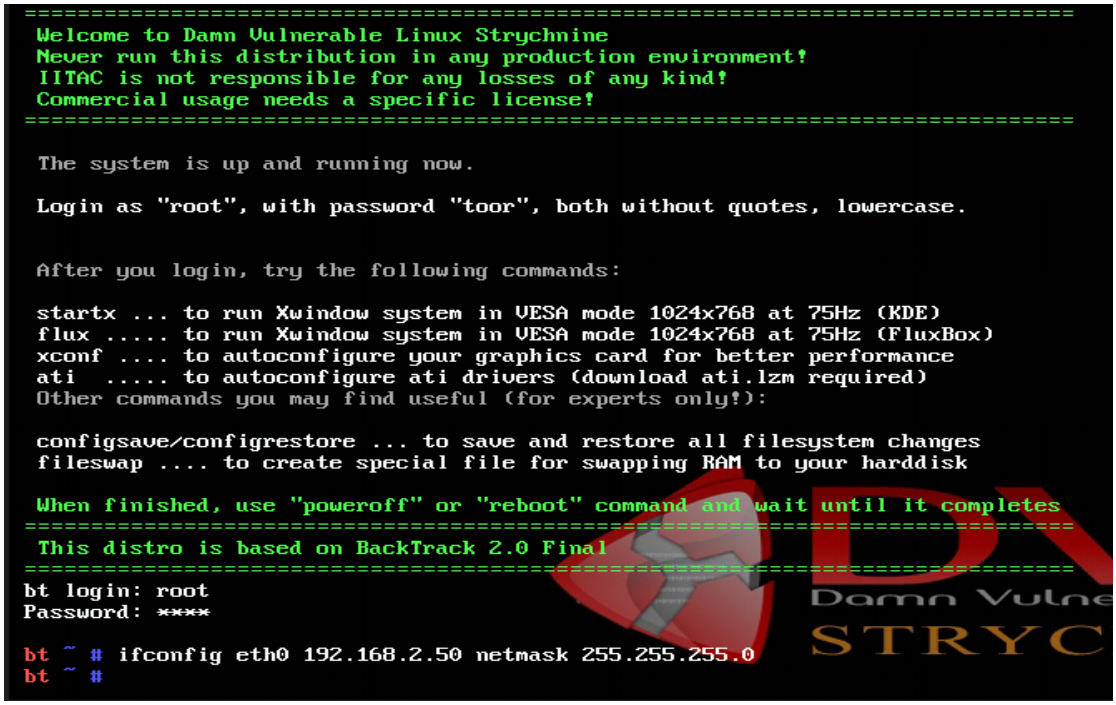

Ilustración 92: Secuencia de comandos de arranque.

Nombre	Creación de rutas
ID	CP 47
Descripción	A continuación, el *script* de salida se encargará de ir configurando las diferentes rutas necesarias para obtener conectividad. Podemos observar esto en la *ilustración 93*, en la cual se observa la creación de 2 rutas.

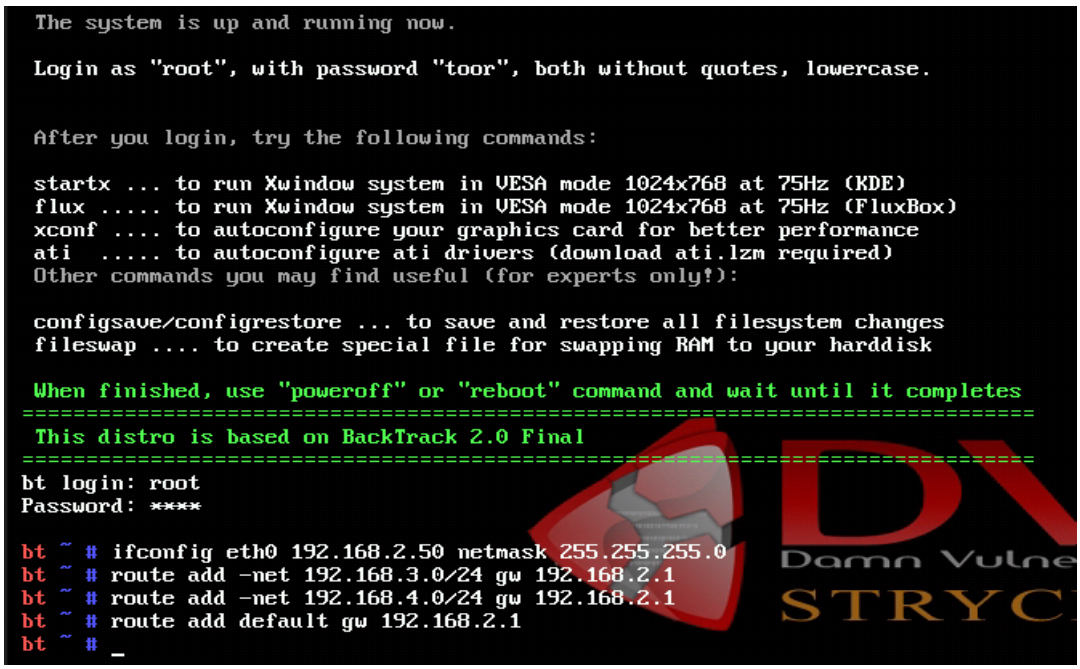

Ilustración 93: Creación de rutas.

Nombre	Lanzamiento de servicios
ID	CP 48
Descripción	El siguiente paso consiste en lanzar los servicios definidos en cada una de las máquinas. Observamos en la *ilustración 94* como se levanta 1 servicio, el servidor *ssh*.

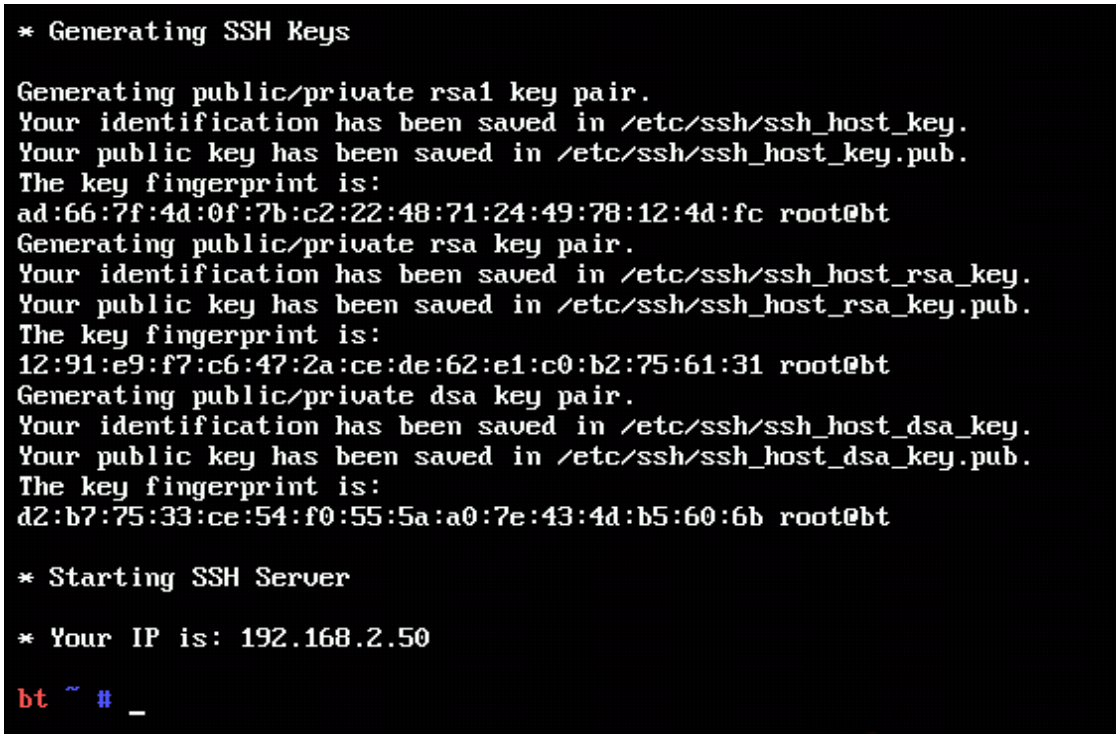

Ilustración 94: Lanzamiento de servicios.

Nombre	Programación de eventos
ID	CP 49
Descripción	El siguiente paso es la programación de los eventos indicados. En la *ilustración 95* podemos ver como se genera un evento ping al *host* con IP 192.168.3.50 para ejecutarse transcurrido 1 minuto.

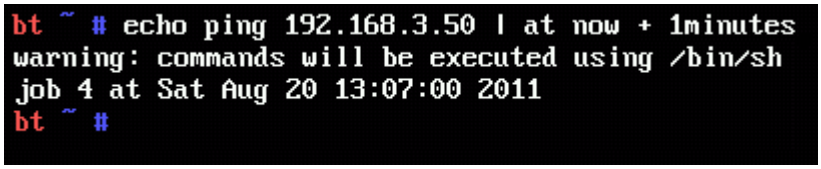

Ilustración 95: Programación de eventos.

Nombre	Creación de reglas para *iptables*
ID	CP 50
Descripción	En paralelo a estos pasos, y en las máquinas que actuan como *gateways*, se configuran las reglas que hacen que estos actuen como *firewalls*. Podemos ver un ejemplo de esto en la *ilustración 96*, donde se crean 2 reglas para *iptables*.

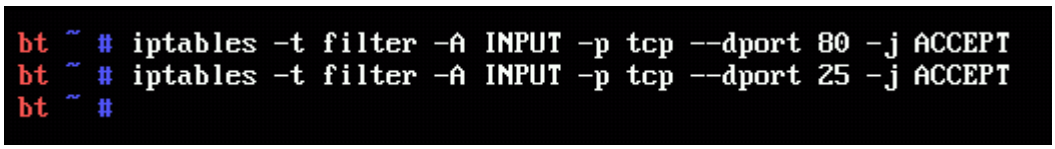

Ilustración 96: Creación de reglas para iptables.

6.2.2. Conectividad

Una vez que se ha realizado el despliegue, se deberá verificar que la configuración ha sido hecha de manera correcta, para lo que se comprobará si hay conectividad entre los diferentes segmentos creados en el escenario.

Nombre	Conectividad
ID	CP 51
Descripción	A continuación se muestra en las *ilustraciones 97,98 y 99* los resultados de hacer un *ping* a los *host* situados en los diferentes segmentos de red, así como los saltos que dan hasta llegar a ellos - comando *traceroute* -, y finalmente la tabla de rutas. Las figuras corresponden a las máquina situadas en el segmento 1, segmento 2 y segmento 3 respectivamente.

Ilustración 97: Pruebas de conectividad desde host situado en el segmento 1.

Ilustración 98: Pruebas de conectividad desde host situado en el segmento 1.

Ilustración 99: Pruebas de conectividad desde host situado en el segmento 3.

6.3. Servicios y eventos

En este apartado se van a realizar las pruebas relacionadas con la verificación de los servicios levantados y los eventos creados.

En primer lugar comprobaremos que los *servicios* definidos en el escenario han sido levantados exitosamente en la máquina virtual correspondiente, y está a la escucha de conexiones entrantes.

A continuación se verificará que los *eventos* han sido creados y están a al espera de ser lanzados en los *host* correspondientes.

6.3.1. Servicios

Como vimos en apartados previos, en la fase de despliegue se lanzaron los servicios definidos en el escenario de pruebas. En este apartado vamos a verificar que los servicios están corriendo, y están a la escucha de conexiones entrantes.

Nombre	Servicios corriendo
ID	CP 52
Descripción	Esto lo podemos observar en las *ilustraciones 100 y 101*, donde en primer lugar se ve que hay un *socket* esperando

	conexiones entrantes de red, y a continuación los procesos asociados con cada servicio. Ambas figuras corresponden a las máquinas situadas en el segmento 1 y el segmento 2 respectivamente, en los cuales se definieron estos servicios.

```
bt ~ # netstat -aveptn
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       User       Inode      PID/Program name
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN      0          26121      19538/X
tcp        0      0 0.0.0.0:631             0.0.0.0:*               LISTEN      0          17447      3623/cupsd
tcp6       0      0 :::6000                 :::*                    LISTEN      0          26120      19538/X
tcp6       0      0 :::22                   :::*                    LISTEN      0          25857      19452/sshd
bt ~ # ps aux | grep --color sshd
root     19452  0.0  0.2   3692  1060 ?        Ss   15:06   0:00 /usr/sbin/sshd
root     23034  0.0  0.0   1668   492 tty2     R+   15:18   0:00 grep --color sshd
bt ~ # _
```

Ilustración 100: Servidor ssh.

```
bt ~ # netstat -aveptn
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       User       Inode      PID/Program name
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN      0          27742      18606/X
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      0          25028      18019/httpd
tcp        0      0 0.0.0.0:631             0.0.0.0:*               LISTEN      0          17392      3621/cupsd
tcp6       0      0 :::6000                 :::*                    LISTEN      0          27741      18606/X
bt ~ # ps aux | grep --color httpd
root     18019  0.0  0.3   4416  1808 ?        Ss   14:59   0:00 /usr/local/apache/bin/httpd
nobody   18021  0.0  0.1   4416  1008 ?        S    14:59   0:00 /usr/local/apache/bin/httpd
nobody   18022  0.0  0.1   4416  1008 ?        S    14:59   0:00 /usr/local/apache/bin/httpd
nobody   18023  0.0  0.1   4416  1008 ?        S    14:59   0:00 /usr/local/apache/bin/httpd
nobody   18024  0.0  0.1   4416  1008 ?        S    14:59   0:00 /usr/local/apache/bin/httpd
nobody   18025  0.0  0.1   4416  1008 ?        S    14:59   0:00 /usr/local/apache/bin/httpd
root     24182  0.0  0.0   1664   488 tty2     R+   15:19   0:00 grep --color httpd
bt ~ #
```

Ilustración 101: Servidor HTTP.

6.3.2. Eventos

Del mismo modo que con los servicios, se ha de verificar que los eventos creados en el escenario de prueba desarrollado para este capítulo sean correctos.

Nombre	Eventos corriendo (cola de eventos)
ID	CP 53
Descripción	En primer lugar vamos a observar como mediante el comando '*at*', usado para generar estos eventos, se hace una lista de los que se encuentran encolados, de esta forma verificaremos que los eventos fueron creados exitosamente. En la *ilustración 102*, podemos observar como efectivamente existe el evento creado en la máquina virtual 1, que corresponde a un ping, como ya se indicó caso de prueba 49.

```
bt ~ # atq
1       Sat Aug 20 13:07:00 2011 a root
bt ~ # _
```

Ilustración 102: Eventos corriendo (cola de eventos).

6.4. Gestión

Finalmente, dedicaremos un breve apartado para realizar las pruebas relacionadas con la gestión. Como ya se indicó, el *software* elegido para realizar la gestión es *virt-manager*, el cual nos provee de una *GUI* (*Graphic User Interface*, Interfaz Gráfica de Usuario) que nos facilita el manejo de todas las máquinas virtuales indicadas en el despliegue.

Nombre	Creación de conexiones en *virt-manager*
ID	CP 54
Descripción	En primer lugar debemos de crear las conexiones a cada máquina anfitriona en el laboratorio, para lo cual debemos seleccionar '*File → Add Connection*', momento en el que nos aparecerá la pantalla que observamos en la *ilustración 103*. Como se puede ver, debemos elegir como tipo de conexión *'Remote tunnel over SSH'*, y en la caja de texto *Hostname*, indicar el usuario y la IP de la máquina que queremos gestionar. Una vez hecho esto, y después de presionar el botón '*Connect*', se nos pedirán las credenciales, en nuestro caso simplemente debemos introducir la contraseña para desbloquer el certificado.

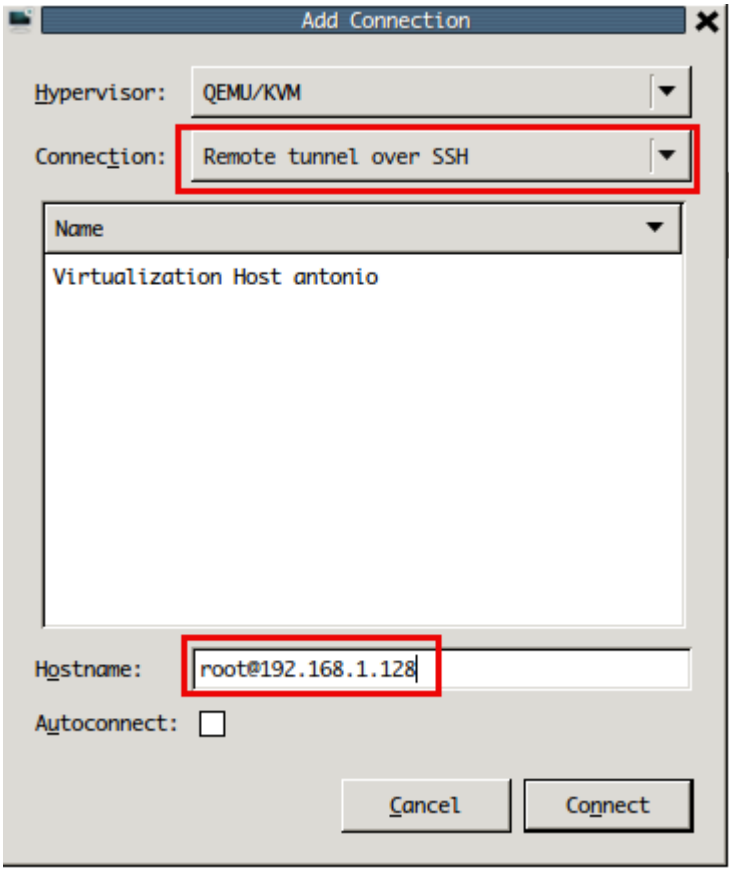

Ilustración 103: Creación de conexión en virt-manager.

Nombre	Creación de conexiones en *virt-manager*
ID	CP 55
Descripción	Como resultado de esto, tendremos en única consola y de manera centralizada, acceso a todas las máquinas virtuales del laboratorio, como se observa en la *ilustración 104*.

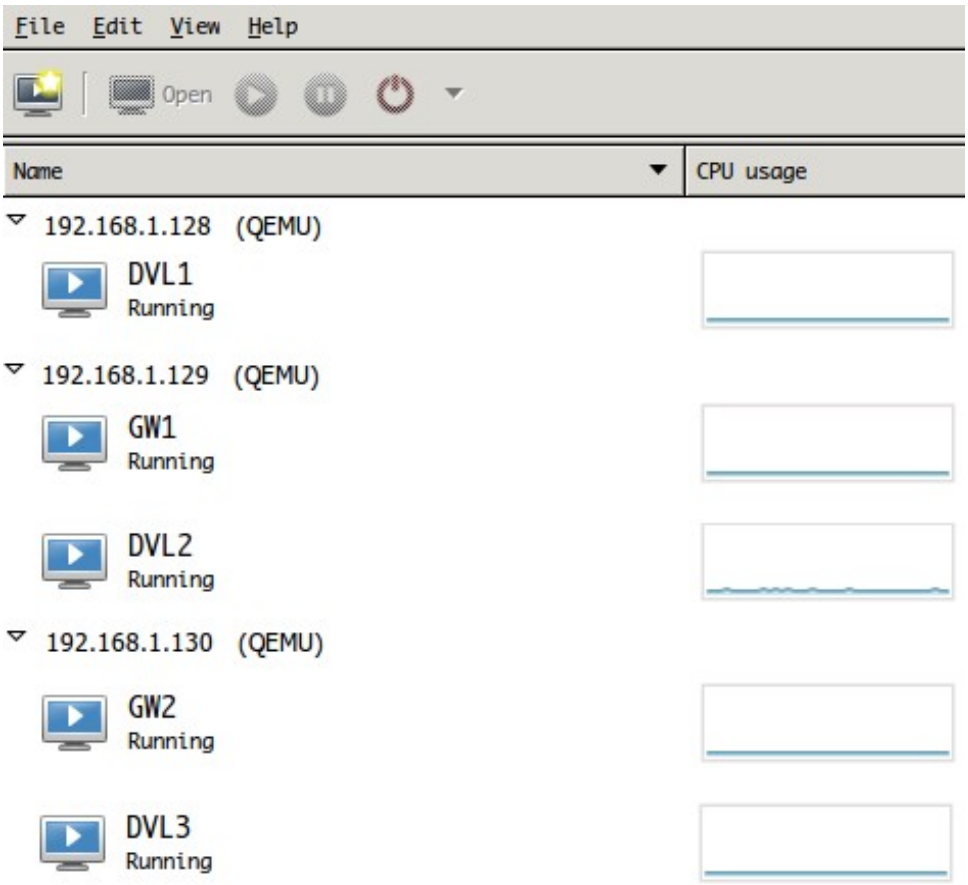

Ilustración 104: Gestión de las máquinas virtuales en virt-manager.

Una vez creadas todas las conexiones, únicamente debemos de hacer doble *click* en una de las máquinas virtuales para acceder mediante *VNC* a una sesión gráfica en ella.

7

CONCLUSIONES Y TRABAJO FUTURO

El punto de partida de este Trabajo de Fin de Carrera fue la necesidad de crear una plataforma sobre la cual basar los laboratorios de las asignaturas de seguridad impartidas en la universidad. Esta necesidad surge debido a que hasta el momento en estos laboratorios las diferentes prácticas que debían superar los alumnos no tenían una cohesión fuerte entre ellas, además de que el parecido con los escenarios y métodos usados en un entorno real - no universitario - eran mínimos. Una vez focalizada la temática del proyecto, se hizo un cuidadoso estudio de la factibilidad del mismo y los requisitos necesarios a cumplir, lo cual se utilizó para comprender los distintos componentes involucrados en el desarrollo de la plataforma, llegando a la conclusión de que los pilares básicos iban a ser los siguientes:

- Virtualización
- *Software* Libre
- Seguridad

Para poder desplegar la plataforma esbozada en el estudio previo, se hace imprescindible el uso de la tecnología de virtualización, la cual nos brinda una potencia y flexibilidad necesaria en el proyecto. Mediante el uso de esta tecnología se nos posibilita el despliegue de distintas máquinas virtuales a lo largo de la red de laboratorio de una forma totalmente configurable, y en el momento que dejen de ser útiles su borrado es sumamente sencillo. De entre todas las opciones existentes, se eligió *KVM* debido a que tras probar con otras plataformas, esta era la que más se acercaba a nuestras necesidades, además de estar fuertemente integrada en las últimas versiones de *Ubuntu*. Este sistema operativo ha sido el usado para todas las máquinas, ya que a parte de haberlo usado durante más de 7 años y por lo tanto tener un profundo conocimiento de él, es una de las distribuciones más fáciles de configurar y está siempre a la vanguardia en cuanto a versiones de *software* en las aplicaciones existentes. Junto a *KVM* se ha usado L*ibvirt*, sin duda uno de los componentes clave de la plataforma desarrollada, ya que gracias a su alta flexibilidad de mediante los ficheros de configuración basados en el formato *XML*, y su modo de trabajo en red *Multicast* ha permitido llevar a buen puerto el proyecto.

Aunque tenía ciertos conocimientos previos en el campo de la virtualización, debido al uso intensivo que he hecho de esta tecnología a lo largo de todo el Trabajo de Fin de Carrera, he adquirido nuevos y más asentados conocimientos, que sin duda me serán útiles de cara a un futuro, y me han servido para descubrir una tecnología altamente compleja pero apasionante, que está llamada a ser básica en un futuro cercano, si es que no lo es ya.

Tanto *KVM*, como *Ubuntu* y L*ibvirt* son aplicaciones de *software* libre, igual que el resto de las herramientas usadas. Si bien a veces no es fácil encontrar aplicaciones con esta filosofía que nos permitan llevar a cabo las tareas requeridas, siempre he conseguido encontrar a lo largo de este Trabajo de Fin de Carrera herramientas libres y alternativas a algunas ya establecidas casi como *standard* en el campo de la informática, como es el caso del sistema operativo *Microsoft Windows*, la plataforma de virtualización *VmWare*, la suite ofimática *Microsoft Office*, y un largo etcétera. A pesar de las dificultades ya citadas, al final las elecciones hechas en cada momento me han brindado la misma o incluso mayor funcionalidad que las herramientas comerciales a un coste nulo y permitiéndome mantener una filosofía en la que creo fuertemente. Algunas de las herramientas se *software* libre usadas a lo largo de este Trabajo de Fin de Carrera eran completamente desconocidas para mi, llevándome una grata sorpresa con todas, pero especialmente con la *API* de la ya mentada *Libvirt*, la cual considero que es sumamente potente y flexible.

En cuanto al trabajo futuro, queda un largo camino por recorrer, pero creo que el trabajo iniciado en este proyecto ha servido para establecer los cimientos de una base sólida que hay que desarrollar y complementar adecuadamente. Bajo mi percepción, los puntos fuertes de la plataforma desarrollada en este Trabajo de Fin de Carrera son la *flexibilidad,* la *extensibilidad,* la *facilidad* de cara al usuario final y la *automatización*. Flexibilidad en cuanto que se pueden definir escenarios completamente diferentes y complejos, en donde los elementos definidos interactuan entre si de la misma forma que lo harían en un escenario real con elementos físicos y no virtuales. Extensibilidad, dado que aunque las pruebas se han hecho con un conjunto reducido de sistemas operativos y plantillas, la forma de definir estos permite que añadir nuevos sistemas operativos y plantillas sea sumamente sencillo, minimizando el esfuerzo necesario para hacer que la plataforma llegue a cubrir un amplio dominio. Facilidad, ya que los profesionales que usen esta plataforma sólo deben preocuparse de crear los escenarios que consideren necesarios para los laboratorios de las asignaturas impartidas en un archivo *XML* con una sintaxis limpia y clara. Y automatización en cuanto que todo el trabajo de realizar el despliegue en los diferentes equipos físicos de un laboratorio, para después encargarse de su correcta configuración, es hecho de manera transparentemente por la plataforma desarrollada.

Partiendo de estos 4 puntos, a continuación voy a dar una serie de opiniones sobre aspectos a pulir en la aplicación, y elementos que añadir en futuros desarrollos.

- Las pruebas hechas en este proyecto han sido realizadas en su totalidad con sistemas operativos en formato LiveCD. Aunque este formato ofrece una gran flexibilidad, sería interesante poder trabajar con instalaciones de sistemas operativos instalados de la manera tradicional, ya que el abanico de sistemas operativos con los que se podría trabajar crecería enormemente.

- En caso de tener las licencias pertinentes, también sería interesante poder trabajar con sistemas operativos propietarios como son *Microsoft Windows*, o *OS* de *Apple*. Esto ofrecería a los alumnos el poder trabajar con sistemas operativos que son ampliamente utilizados en entornos reales.

- Debido a que la comunicación con las máquinas virtuales se hace mediante el *script* desarrollado en *Python*, basándose en el protocolo *RBN*, nos encontramos con ciertas dificultades a la hora de manejar los mapas de caracteres, más en concreto los

diferentes mapas existentes para cada idioma. Por ejemplo, si se configura el *script* para que trabaje con el mapa de caracteres español, y la máquina por defecto trabaja con el mapa americano, nos encontramos con que ciertos códigos de caracter serán interpretados de manera diferente. Sería importante pensar en una solución a este problema para facilitar la tarea de añadir nuevos sistemas operativos sin que el mapa de caracteres sea un problema, o alternativamente, diseñar otra solución a la hora de comunicarse con las máquinas virtuales.

- Dado a la necesidad de separar las máquinas en las cuales se despliega el escenario de las máquinas usadas para que los alumnos trabajen, sería interesante diseñar algún mecanismo que permita hacer esto de manera automática.

- Hasta el momento, el único tipo de evento definido son los ataques, pero debido a la sintaxis sencilla y extensible a la hora de definir plantillas y escenarios, sería interesante diseñar nuevos tipos de eventos que amplíen la variedad y funcionalidad inicial de los eventos.

- Dado el flujo actual de trabajo, una vez que el profesor ha definido y desplegado el escenario, para poder gestionar las máquinas tiene que acceder directamente a una consola en ellas y realizar a mano las tareas que desee. Sería interesante introducir una nueva funcionalidad que permita gestionar en tiempo real y dinámicamente los escenarios de una forma similar a la hecha en la definición de este, sin la necesidad de acceder a las máquinas virtuales implícitamente.

- Una mejora de cada a la usabilidad del usuario final, sería el desarrollo de una *GUI (Graphic User Interface*, Interfaz Gráfica de Usuario*)* que encapsulara todas las funcionalidades ofrecidas por el proyecto en un entorno gráfico.

- Además del punto anterior, sería un añadido el poder definir los escenarios y las plantillas gráficamente, de forma que el uso de archivos *XML* fuera transparente para el usuario final.

- La instalación y configuración inicial de los diferentes componentes a lo largo de todo el laboratorio se debe hacer a mano, por lo que sería interesante automatizar esta tarea mediante un *script*, que se encargase de todas las tareas a hacer en la fase inicial.

Antes de finalizar, indicar que a pesar de estas sugerencias de cara al futuro, los requisitos de los que se partió a la hora de definir el alcance de este Trabajo de Fin de Carrera, han sido cumplidos, siendo estos puntos descritos anteriormente ideas que sobre los pilares construidos ayudarán a mejorar y ampliar las funcionalidades del proyecto.

BIBLIOGRAFÍA

alt1040.com (2010), "Stuxnet y el sombrío 9 de Mayo de 1979", (http://alt1040.com/2010/10/stuxnet-y-el-sombrio-9-de-mayo-de-1979)

AMD (2011), "AMD Virtualization", (http://sites.amd.com/us/business/it-solutions/virtualization/Pages/virtualization.aspx)

André B. Bondi (2000), "Characteristics of scalability and their impact on performance", Proceedings of the 2nd international workshop on Software and performance, Ottawa, Ontario, Canada.

Daniel J. Barrett,Richard E. Silverman,Robert G. Byrnes (2005), "SSH, the secure shell: the definitive guide", O'Reilly.

Dragon Jar (2010), "The Social-Engineer Toolkit", (http://www.dragonjar.org/the-social-engineer-toolkit.xhtml)

El Economista (2010), "Operación Aurora, el ciberataque más sofisticado de la historia", (http://eleconomista.com.mx/tecnociencia/2010/01/20/operacion-aurora-ciberataque-mas-sofisticado-historia)

Enrique de la Hoz, Iván Marsá-Maestre, Miguel A. Lopez-Carmona, María Teresa López-Merayo (2010), " Herramienta para la generación de escenarios de apoyo a la docencia de la seguridad en redes.", Actas de las I Jornadas de Innovación Educativa en Ingeniería Telemática (JIE/JITEL 2010)

Eroski Consumer (2005), "Virus Sober", (http://www.consumer.es/web/es/tecnologia/2005/05/04/141699.php)

ESET (2009), "El arma inefable: la Ingeniería Social", (http://www.eset-la.com/press/informe/arma_infalible_ingenieria_social.pdf)

Free Software Foundation (2009), "The Free Software Definition" (http://www.gnu.org/philosophy/free-sw.html)

GNU(2009), “Definición Software Libre”, (http://www.gnu.org/philosophy/free-sw.es.html)

GNU (2009), "¿Qué es el copyleft?" (http://www.gnu.org/copyleft/copyleft.es.html)

HP (2011), "HP Integrity Virtual Machines", (http://h20271.www2.hp.com/enterprise/us/en/os/hpux11i-partitioning-integrity-vm.html)

IBM Research (2009), "Michelangelo Madness" (http://www.research.ibm.com/antivirus/SciPapers/White/VB95/vb95.distrib-node7.html)

IEET (2007), "Storm Botnet storms the net"

(http://ieet.org/index.php/IEET/more/dvorsky20070927/)

IETF (2006), “The Secure Shell (SSH) Protocol Architecture”, (http://www.ietf.org/rfc/rfc4251.txt)

InfoSpyware (2009), "¿Qué son los Malwares?", (http://www.infospyware.com/articulos/que-son-los-malwares/)

Irongeek (2011), "Programmable HID USB Keystroke Dongle: Using the Teensy as a pen testing device (Defcon 18)", (http://www.irongeek.com/i.php?page=videos/phukd-defcon-18)

Japan Business People technology (2007), "Types of Virtualization", (http://www.japaninc.com/mgz_nov-dec_2007_virtualization)

J. B. G. Randal T. Abler, Didier Contis and H. L. Owen (2006), "Georgia tech information security center hands-on network security laboratory", IEEE Transactions on Education.

María Teresa Jimeno García, Carlos Míguez Pérez, Abel Mariano Matas García, Justo Pérez Agudín (2008), "*Hacker"*, Anaya.

J. Smith and R. Nair (2005), "The architecture of virtual machines", Computer.

Libvirt (2011), "Format Domain", (http://*libvirt.org/formatdomain.html*)

McAfee (2011), "Predicciones de amenazas para 2011", (www.mcafee.com/es/resources/reports/rp-threat-predictions-2011.pdf)

Menken, Ivanka (2010), "Virtualization - The Complete Cornerstone Guide to Virtualization Best Practices", EMEREO PTY LTD; 2 Exp Upd edition.

Metasploit, http://www.metasploit.com/

MÉTRICA v.3., Ministerio de Administraciones Públicas de España.

Microsoft (2010), "CVE-2010-0249", (http://www.microsoft.com/technet/security/advisory/979352.mspx)

Open Source Initiative (2011), "The Open Source Definition", (http://www.opensource.org/docs/osd)

Pedro Tortosa (2008), "El futuro de la gestión de sistemas de TI se llama virtualización ", Datamation.

Sommerville, Ian (2006). "Software Engineering”, (8th ed.)

Python (2008), "Python Library Reference",

(http://docs.python.org/release/2.5.2/lib/lib.html)

QEMU (2011), http://wiki.qemu.org/Main_Page
Intel (2011), "Virtualization",
(http://www.intel.com/technology/virtualization/)

ReAssure (2010), Web oficial,
(http://projects.cerias.purdue.edu/reassure/)

RFB v.3.8. (2009), "RFB Protocol"
http://www.realvnc.com/docs/rfbproto.pdf

SET (2010), "Social Engineering Toolkit",
(http://www.social-engineer.org/framework/Computer_Based_Social_Engineering_Tools:_Social_Engineer_Toolkit_%28SET%29)

Stallings, W. (2008), "Operating Systems. Internals and Design Principles", 6th ed. Pearson.

Stevens, Perdita y Pooley, Rob (2007), "Utilización De Uml en Ingenieria del Software con Objetos y Componentes". Addison and Wesley.

Symantec (2010), "A history of viruses",
(http://www.symantec.com/connect/articles/history-viruses)

Symantec (2002), "W95.CIH",
(http://www.symantec.com/security_response/writeup.jsp?docid=2000-122010-2655-99)

Symantec (2008), "Symantec Report on the Internet Underground Economy",
(http://eval.symantec.com%2Fmktginfo%2Fenterprise%2Fwhite_papers%2Fb-whitepaper_underground_economy_report_11-2008-14525717.en-us.pdf)

T. A. Yang, K.-B. Yue, M. Liaw, G. Collins, J. T. Venkatraman,
S. Achar, K. Sadasivam, and P. Chen (2004), "Design of a distributed computer security lab". 8th Austalian conference on Computing education.

UAH (2008)," Programa Seguridad en Internet",
(http://it.aut.uah.es/enrique/docencia/ii/seguridad/documentos/programa-0506.pdf)

Virus Scan Software (2011), "The history of computer viruses",
(http://www.virus-scan-software.com/virus-scan-help/answers/the-history-of-computer-viruses.shtml)

Virus Wikia (2010), "Stoned",
(http://virus.wikia.com/wiki/Stoned)

Vmware (2007), "Understanding Full Virtualization, Paravirtualization, and Hardware Assisted Virtualization", (http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf)

Vsantivirus (2003), "W32/Lovsan.A (Blaster). Utiliza la falla en RPC", (http://www.vsantivirus.com/lovsan-a.htm)

Warzone (2007), “Wargame 1.0”, (http://warzone.elhacker.net/)

Web Sense (2011), "2010 Threar Report", (http://www.dsci.in/sites/default/files/report-websense-2010-threat-report-en.pdf)

Wikipedia (2011), "Ingeniería de Requisitos", (http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_requisitos)

Wikipedia (2011), "Ingeniería social", (http://es.wikipedia.org/wiki/Ingenier%C3%ADa_social_%28seguridad_inform%C3%A1tica%29)

Wikipedia (2009), "Virus Informático", (http://es.wikipedia.org/wiki/Virus_inform%C3%A1tico#Historia)

Xen (2008), "Quantitative Comparison of Xen and KVM ", (http://wiki.xensource.com%2Fxenwiki%2FOpen_Topics_For_Discussion%3Faction%3DAttachFile%26do%3Dget%26target%3DQuantitative%2BComparison%2Bof%2BXen%2Band%2BKVM.pdf)

ZHAO Fang-fang,CHEN Xiu-zhen,LI Jian-hua (2008, "Generation Mehtod of Network Attack Graphs Based on Privilege Escalation", School of Electronic Information and Electrical Engineering,Shanghai Jiaotong University,Shanghai.

ANEXO A – Código Fuente

Archivo '*pfc.py*'

```
import sys                                      #Importamos libreria sys
from optparse import OptionParser               #Importamos libreria para parsear argumentos de
linea de comandos
import re                                       #Importamos libreria para expresiones regulares
import ConfigParser                             #Importamos libreria para parsear archivo de
configuracion
try:
  from lxml import etree                #Importamos libreria para parsear documentos XML
except ImportError:
  import xml.etree.ElementTree as etree




class Template_parser:
        '''Clase que parsea y valida la definicion de una Plantilla'''


        def __init__(self, template_path):
                self.template_path = template_path


                try:
                        self.doc = etree.parse (self.template_path)    #Abrimos archivo de
escenario
                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing XML
file: %s\n\n" % str(e)
                sys.exit(1)


        #def __del__(self):


        def __parse_vm(self, root):
                if (not(root.attrib.get('name') and root.attrib.get('path') and
root.attrib.get('template') and root.attrib.get('boot'))):
                                print "\n\n\t[+] Error: Vm tag has no NAME or PATH or TEMPLATE
or BOOT parameters\n\n"
                                sys.exit(1)
                for child in root:
                        if (child.tag == 'service'):
                                gateway = self.__parse_service(child)
                        elif (child.tag == 'attack'):
                                host = self.__parse_attack(child)
```

```
                        else:
                                print "\n\n\t[+] Error: Tag element different from <SERVICE> or
<ATTACK>\n\n"
                                sys.exit(1)

        def __parse_service(self, root):
                if (not(root.attrib.get('name') and root.attrib.get('init'))):
                                print "\n\n\t[+] Error: Service tag has no NAME or INIT
parameters\n\n"
                                sys.exit(1)

                for child in root:
                        if (child.tag == 'param'):
                                gateway = self.__parse_param(child)
                        else:
                                print "\n\n\t[+] Error: Tag element different from <PARAM>\n\n"
                                sys.exit(1)

        def __parse_attack(self, root):
                if (not(root.attrib.get('name') and root.attrib.get('command'))):
                                print "\n\n\t[+] Error: Attack tag has no NAME or COMMAND
parameters\n\n"
                                sys.exit(1)

                for child in root:
                        if (child.tag == 'type'):
                                gateway = self.__parse_type(child)
                        else:
                                print "\n\n\t[+] Error: Tag element different from <TYPE> \n\n"
                                sys.exit(1)

        def __parse_param(self, root):
                        if (not(root.attrib.get('name') and root.attrib.get('mod'))):
                                print "\n\n\t[+] Error: Param tag has no NAME or MOD(empty?->
use "") parameters\n\n"
                                sys.exit(1)
        def __parse_type(self, root):
                if (not(root.attrib.get('name'))):
                                print "\n\n\t[+] Error: Type tag has no NAME parameter\n\n"
                                sys.exit(1)

                for child in root:
                        if (child.tag == 'param'):
                                gateway = self.__parse_param(child)
                        else:
```

```
                print "\n\n\t[+] Error: Tag element different from <PARAM>\n\n"
                sys.exit(1)

        def parse_template(self):
                root = self.doc.getroot()              #Cogemos el elemento raiz del escenario

                if (root.tag != 'templates'):
                        print "\n\n\t[+] Error: Root element different from <TEMPLATES>\n\n"
                        sys.exit(1)

                for child in root:
                        if (child.tag == 'vm'):
                                segment = self.__parse_vm(child)
                        else:
                                print "\n\n\t[+] Error: Tag element different from <VM>\n\n"
                                sys.exit(1)

class Stage_parser:
        '''Clase que implementa el parser del escenario'''

        def __init__(self, stage_path, template_path):
                self.stage_path = stage_path
                self.template_path = template_path

                try:
                        self.doc = etree.parse (self.stage_path)       #Abrimos archivo de
escenario
                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing XML
file: %s\n\n" % str(e)
                sys.exit(1)


        #def __del__(self):


        def __parse_stage(self, doc):
                root = doc.getroot()            #Cogemos el elemento raiz del escenario

                if (root.tag != 'stage'):
                        print "\n\n\t[+] Error: Root element different from <STAGE>\n\n"
                        sys.exit(1)

                stage = Stage()                 #Creamos una instancia de la Clase Stage
```

```
                for child in root:
                        if (child.tag == 'segment'):
                                segment = self.__parse_segment(child)
                                stage.add_segment(segment)            #Aniadimos el segmento
parseado a la lista de segmentos
                        else:
                                print "\n\n\t[+] Error: Tag element different from
<SEGMENT>\n\n"
                                sys.exit(1)
                return stage                            #Devolvemos el stage


        def __parse_segment(self, root):
                segment = Segment(root.attrib)          #Creamos una instancia de la Clase
Segment


                vnc_count = 0
                for child in root:
                        if (child.tag == 'gateway'):
                                gateway = self.__parse_gateway(child, vnc_count)
                                segment.add_gateway(gateway)  #Aniadimos el gateway parseado a
la lista de gateways
                        elif (child.tag == 'host'):
                                host = self.__parse_host(child, vnc_count)
                                segment.add_host(host)        #Aniadimos el host parseado a la
lista de hosts
                        else:
                                print "\n\n\t[+] Error: Tag element different from <HOST> or
<GATEWAY>\n\n"
                                sys.exit(1)


                        vnc_count += 1


                return segment                                  #Devolvemos el segmento




        def __parse_host(self, root, vnc_display):
                host = Host(root.attrib, vnc_display, self.template_path)            #Creamos una
instancia de la Clase Host


                for child in root:
                        if (child.tag == 'route'):
                                route = self.__parse_route(child)
                                host.add_route(route)                #Aniadimos la ruta
parseada a la lista de rutas
                        elif (child.tag == 'service'):
```

```
                                service = self.__parse_service(child)

                                (exist, params) = service.check_exists(self.template_path,
root.attrib['template'])
                                if (not exist or not params):           #Comprobamos que existe el
servicio
                                        print "\n\n\t[+] Error: Service Name:",
child.attrib['name'], "in Template Name:", root.attrib['template'],\
                                        "does not exist or type/parameters are wrong!\n\n"
                                        sys.exit(1)

                                host.add_service(service)               #Aniadimos el servicio
parseado a la lista de servicios
                        elif (child.tag == 'event'):
                                event = self.__parse_event(child)

                                for attack in event.get_attacks():
                                        (exist, typ, params) =
attack.check_exists(self.template_path, root.attrib['template'])    #Comprobamos que existe el
ataque, tipo y parametros
                                        if (not exist or not typ or not params):
                                                print "\n\n\t[+] Error: Attack Name:",
attack.template, "in Template Name:",\
                                                root.attrib['template'], "does not exist or
type/parameters are wrong!\n\n"
                                                sys.exit(1)

                                host.add_event(event)                   #Aniadimos el evento
parseado a la lista de eventos
                        else:
                                print "\n\n\t[+] Error: Tag element different from <ROUTE>,
<SERVICE> or <EVENT>\n\n"
                                sys.exit(1)

                return host                                     #Devolvemos el host


        def __parse_gateway(self, root, vnc_display):
                gateway = Gateway(root.attrib, vnc_display, self.template_path)
        #Creamos una instancia de la Clase Gateway

                for child in root:
                        if (child.tag == 'route'):
                                route = self.__parse_route(child)
                                gateway.add_route(route)                #Aniadimos la ruta
parseada a la lista de rutas
                        elif (child.tag == 'ipchains'):
                                ipchain = self.__parse_ipchain(child)
```

```
                                gateway.add_ipchain(ipchain)          #Aniadimos la cadena
ipchain parseada a la lista de ipchains
                        else:
                                print "\n\n\t[+] Error: Tag element different from <ROUTE> or
<IPCHAINS>\n\n"
                                sys.exit(1)

                return gateway                                  #Devolvemos el gateway


        def __parse_route(self, root):
                route = Route(root.attrib)              #Creamos una instancia de la Clase Route

                return route                            #Devolvemos la ruta

        def __parse_service(self, root):
                service = Service(root.attrib)          #Creamos una instancia de la Clase
Service
                return service                          #Devolvemos el servicio

        def __parse_event(self, root):
                event = Event(root.attrib)              #Creamos una instancia de la Clase Event


                for child in root:
                        if (child.tag == 'attack'):
                                attack = self.__parse_attack(child)

                                event.add_attack(attack)              #Aniadimos el ataque
parseado a la lista de ataques
                        else:
                                print "\n\n\t[+] Error: Tag element different from <ATTACK>\n\n"
                                sys.exit(1)

                return event                                    #Devolvemos el evento


        def __parse_attack(self, root):
                attack = Attack(root.attrib)            #Creamos una instancia de la Clase Attack

                return attack                           #Devolvemos el ataque

        def __parse_ipchain(self, root):
                ipchain = Ipchain(root.attrib)          #Creamos una instancia de la Clase
Ipchain
                return ipchain                          #Devolvemos la regla ipchain
```

```
        def parse_xml(self):
                stage = self.__parse_stage(self.doc)
                return stage


class Stage:
        '''Clase que implementa un escenario'''

        def __init__(self):
                self.segments = []

        def add_segment(self, segment):
                self.segments.append(segment)

        def print_stage(self):
                for segment in self.segments:
                        segment.print_segment()

        def configure_stage(self, output_file, vnc_sendkeys):
                try:
                fd = open(output_file, 'w')     #abrimos el archivo en modo append, donde se
guardara el script de salida
                except IOError:
                print "\n\n\t[+] An error was encountered while opening %s file" % output_file
                else:
                        last_segment_mcast_port = 0
                        for segment in self.segments:
                                segment.configure_segment(fd, vnc_sendkeys,
last_segment_mcast_port)
                                last_segment_mcast_port = segment.get_port()
                        fd.close()

        #def __del__(self):

class Segment:
        '''Clase que implementa un segmento de red'''

        def __init__(self, attrs):
                try:
                        self.label = attrs['label']
                        self.host = attrs['host']
                        self.mcast = attrs['mcast']
                        self.port = attrs['port']
                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing
<SEGMENT> tag: %s\n\n" % str(e)
```

```
                sys.exit(1)

                self.hosts = []
                self.gateways = []

        def add_host(self, host):
                self.hosts.append(host)

        def add_gateway(self, gateway):
                self.gateways.append(gateway)

        def get_port(self):
                return self.port

        def print_segment(self):
                print "SEGMENT:\n", self.label, " ", self.host, " ", self.mcast, " ", self.port
                for gateway in self.gateways:
                        gateway.print_gateway()
                for host in self.hosts:
                        host.print_host()

        def configure_segment(self, fd, vnc_sendkeys, mcast_link_port):
                try:
                        script_line = '\nssh root@%s \'\n' % self.host

                        fd.write(script_line)
                        #otras confs
                        for gateway in self.gateways:
                                gateway.create_gw(fd, self.mcast, mcast_link_port, self.port)
                        for host in self.hosts:
                                host.create_host(fd, self.mcast, self.port)

                        fd.write('\'\n\n')

                        for gateway in self.gateways:
                                gateway.configure_gw( fd,vnc_sendkeys, self.host)
                        for host in self.hosts:
                                host.configure_host( fd, vnc_sendkeys, self.host)

                        for gateway in self.gateways:
                                gateway.configure_routes( fd, vnc_sendkeys, self.host)
                                gateway.configure_ipchains(fd, vnc_sendkeys, self.host)
                        for host in self.hosts:
                                host.configure_routes(fd, vnc_sendkeys, self.host)
```

```
                                host.configure_services(fd, vnc_sendkeys, self.host)
                                host.configure_events(fd, vnc_sendkeys, self.host)


                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s
file" % fd



        #def __del__(self):


class Host:
        '''Clase que implementa un host'''


        def __init__(self, attrs, vnc_display, template_path):
                try:
                        self.ip = attrs['ip']
                        self.template = attrs['template']
                        self.name = attrs['name']
                        self.vnc_display = vnc_display
                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing <HOST>
tag: %s\n\n" % str(e)
                sys.exit(1)


                if (not self.check_exists(template_path, self.template)):            #Comprobamos
que existe la template
                        print "\n\n\t[+] Error: Template Name: ", self.template, " does not
exist!\n\n"
                        sys.exit(1)


                self.routes = []
                self.services = []
                self.events = []


                (path, template, boot) = self.get_vm_info(template_path, self.template)
                self.path = path
                self.template = template
                self.boot = boot


        def add_route(self, route):
                self.routes.append(route)


        def add_service(self, service):
                self.services.append(service)
```

```
        def add_event(self, event):
                self.events.append(event)


        def filter_string(self, string):
                pattern = '/'   #Patron a buscar
                p = re.compile(pattern)
                return p.sub('\\/', string)    #Sustituimos / por \/


        def check_exists(self, templates_file, template_name):
                try:
                        doc = etree.parse (templates_file)     #Abrimos archivo de plantillas


                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing XML
file: %s\n\n" % str(e)
                sys.exit(1)


                templates = doc.findall('vm')          #Buscamos todas las VMs definidas


                exist=0
                for template in templates:
                        if (template.attrib['name'] == template_name):
                                exist=1                #VM encontrada
                                break


                return exist           #devolvemos resultado de la busqueda


        def get_vm_info(self, templates_file, template_name):
                try:
                        doc = etree.parse (templates_file)     #Abrimos archivo de plantillas


                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing XML
file: %s\n\n" % str(e)
                sys.exit(1)


                templates = doc.findall('vm')          #Buscamos todas las VMs definidas


                for template in templates:
                        if (template.attrib['name'] == template_name):
                                return
(template.attrib['path'],template.attrib['template'],template.attrib['boot'])


        def create_host(self, fd, mcast, port):
                try:
```

```
                        filtered_path = self.filter_string(self.path)
                        filtered_template = self.filter_string(self.template)


                        script_line = 'cp %s %s.tmp;\n' % (filtered_template, filtered_template)
                        script_line += 'sed -i \"s/%%file_iso%%/%s/g\" %s.tmp;\n' %
(filtered_path, filtered_template)
                        script_line += 'sed -i \"s/%%name%%/%s/g\" %s.tmp;\n' % (self.name,
filtered_template)
                        script_line += 'dd if=/dev/zero of=%s.img bs=512M count=4;\n' %
filtered_template
                        script_line += 'sed -i \"s/%%file_disk%%/%s.img/g\" %s.tmp;\n' %
(filtered_template, filtered_template)
                        script_line += 'sed -i \"s/%%uuid%%/`uuidgen`/g\" %s.tmp;\n' %
filtered_template
                        script_line += 'sed -i \"s/%%mcast%%/%s/g\" %s.tmp;\n' % (mcast,
filtered_template)
                        script_line += 'sed -i \"s/%%port%%/%s/g\" %s.tmp;\n' % (port,
filtered_template)
                        script_line += 'virsh -c qemu:///system create %s.tmp;\n' %
filtered_template
                        script_line += 'rm %s.tmp;\n' % filtered_template
                        fd.write(script_line)
                        #otras confs


                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s
file" % fd


        def configure_host(self, fd, vnc_sendkeys, host_ip):
                try:
                        script_line = 'cp %s %s.tmp;\n' % (self.boot, self.boot)
                        script_line += 'sed -i \"s/%%ip%%/%s/g\" %s.tmp;\n' % (self.ip,
self.template)
                        script_line += 'python %s %s %i %s.tmp\n' % (vnc_sendkeys, host_ip,
(self.vnc_display + 5900), self.boot)
                        script_line += 'rm %s.tmp;\n\n' % self.boot
                        fd.write(script_line)
                        #otras confs


                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s
file" % fd


        def configure_routes(self, fd, vnc_sendkeys, host_ip):
                for route in self.routes:
                        route.configure_route(fd, vnc_sendkeys, host_ip, self.vnc_display)
```

```
                last_route = self.routes[-1]
                last_route.configure_default_gw(fd, vnc_sendkeys, host_ip, self.vnc_display)

        def configure_services(self, fd, vnc_sendkeys, host_ip):
                for service in self.services:
                        service.start_service(fd, vnc_sendkeys, host_ip, self.vnc_display)

        def configure_events(self, fd, vnc_sendkeys, host_ip):
                for event in self.events:
                        event.create_event(fd, vnc_sendkeys, host_ip, self.vnc_display)

        def print_host(self):
                print "\tHOST:\n\t", self.ip, " ", self.template, " ", self.name
                for route in self.routes:
                        route.print_route()
                for service in self.services:
                        service.print_service()
                for event in self.events:
                        event.print_event()

        #def __del__(self):

class Gateway:
        '''Clase que implementa un host'''

        def __init__(self, attrs, vnc_display, template_path):
                try:
                        self.template = attrs['template']
                        self.ipin = attrs['ipin']
                        self.ipout = attrs['ipout']
                        self.linkto = attrs['linkto']
                        self.name = attrs['name']
                        self.vnc_display = vnc_display
                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing
<GATEWAY> tag: %s\n\n" % str(e)
                sys.exit(1)

                if (not self.check_exists(template_path, self.template)):           #comprobamos
que existe la template
                        print "\n\n\t[+] Error: Template Name: ", self.template, " does not
exist!\n\n"
                        sys.exit(1)

                self.routes = []
```

```
                self.ipchains = []

                (path, template, boot) = self.get_vm_info(template_path, self.template)
                self.path = path
                self.template = template
                self.boot = boot

        def add_route(self, route):
                self.routes.append(route)

        def add_ipchain(self, ipchain):
                self.ipchains.append(ipchain)

        def filter_string(self, string):
                pattern = '/'   #Patron a buscar
                p = re.compile(pattern)
                return p.sub('\\/', string)    #Sustituimos / por \/

        def check_exists(self, templates_file, template_name):
                try:
                        doc = etree.parse (templates_file)     #Abrimos archivo de plantillas

                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing XML
file: %s\n\n" % str(e)
                sys.exit(1)

                templates = doc.findall('vm')          #Buscamos todas las VMs definidas

                exist=0
                for template in templates:
                        if (template.attrib['name'] == template_name):
                                exist=1                #VM encontrada
                                break

                return exist           #devolvemos resultado de la busqueda

        def get_vm_info(self, templates_file, template_name):
                try:
                        doc = etree.parse (templates_file)     #Abrimos archivo de plantillas

                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing XML
file: %s\n\n" % str(e)
```

```
                sys.exit(1)

                templates = doc.findall('vm')          #Buscamos todas las VMs definidas

                for template in templates:
                        if (template.attrib['name'] == template_name):
                                return
(template.attrib['path'],template.attrib['template'],template.attrib['boot'])

        def create_gw(self, fd, mcast, port1, port2):
                try:
                        filtered_path = self.filter_string(self.path)
                        filtered_template = self.filter_string(self.template)

                        script_line = 'cp %s %s.tmp;\n' % (filtered_template, filtered_template)
                        script_line += 'sed -i \"s/%%file_iso%%/%s/g\" %s.tmp;\n' %
(filtered_path, filtered_template)
                        script_line += 'sed -i \"s/%%name%%/%s/g\" %s.tmp;\n' % (self.name,
filtered_template)
                        script_line += 'dd if=/dev/zero of=%s.img bs=512M count=4;\n' %
filtered_template
                        script_line += 'sed -i \"s/%%file_disk%%/%s.img/g\" %s.tmp;\n' %
(filtered_template, filtered_template)
                        script_line += 'sed -i \"s/%%uuid%%/`uuidgen`/g\" %s.tmp;\n' %
filtered_template
                        script_line += 'sed -i \"s/%%mcast%%/%s/g\" %s.tmp;\n' % (mcast,
filtered_template)
                        script_line += 'sed -i \"s/%%port%%/%s/g\" %s.tmp;\n' % (port1,
filtered_template)
                        script_line += 'sed -i \"s/%%port2%%/%s/g\" %s.tmp;\n' % (port2,
filtered_template)
                        script_line += 'virsh -c qemu:///system create %s.tmp;\n' %
filtered_template
                        script_line += 'rm %s.tmp;\n' % filtered_template
                        fd.write(script_line)
                        #otras confs

                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s
file" % fd

        def configure_gw(self, fd, vnc_sendkeys, host_ip):
                try:
                        script_line = 'cp %s %s.tmp;\n' % (self.boot, self.boot)
                        script_line += 'sed -i \"s/%%ip%%/%s/g\" %s.tmp;\n' % (self.ipin,
self.boot)
                        script_line += 'sed -i \"s/%%ip2%%/%s/g\" %s.tmp;\n' % (self.ipout,
self.boot)
```

```
                        script_line += 'python %s %s %i %s.tmp\n' % (vnc_sendkeys, host_ip,
(self.vnc_display + 5900), self.boot)
                        script_line += 'rm %s.tmp;\n\n' % self.boot
                        fd.write(script_line)
                        #otras confs

                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s
file" % fd

        def configure_routes(self, fd, vnc_sendkeys, host_ip):
                for route in self.routes:
                        route.configure_route(fd, vnc_sendkeys, host_ip, self.vnc_display)

        def configure_ipchains(self, fd, vnc_sendkeys, host_ip):
                for ipchain in self.ipchains:
                        ipchain.create_rule(fd, vnc_sendkeys, host_ip, self.vnc_display)

        def print_gateway(self):
                print "\tGATEWAY:\n\t", self.template, " ", self.ipin, " ", self.ipout, " ",
self.linkto, " ", self.name
                for route in self.routes:
                        route.print_route()
                for ipchain in self.ipchains:
                        ipchain.print_ipchain()

        #def __del__(self):


class Route:
        '''Clase que implementa una route de red'''

        def __init__(self, attrs):
                try:
                        self.net = attrs['net']
                        self.gw = attrs['gw']
                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing <ROUTE>
tag: %s\n\n" % str(e)
                sys.exit(1)

        def configure_route(self, fd, vnc_sendkeys, host_ip, vnc_display):
                try:
                        script_line = 'echo -e \"send_string route add \'net %s gw %s\\nsend_key
Return\\nq\\n\" | python %s %s %i\n'\
```

```
                                        % (self.net, self.gw, vnc_sendkeys, host_ip, vnc_display +
5900)
                        fd.write(script_line)
                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s
file" % fd


        def configure_default_gw(self, fd, vnc_sendkeys, host_ip, vnc_display):
                try:
                        script_line = 'echo -e \"send_string route add default gw %s\\nsend_key
Return\\nq\\n\" | python %s %s %i\n'\
                                        % (self.gw, vnc_sendkeys, host_ip, vnc_display + 5900)
                        fd.write(script_line)
                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s
file" % fd


        def print_route(self):
                print "\t\tROUTE:\n\t\t", self.net, " ", self.gw


        #def __del__(self):


class Service:
        '''Clase que implementa un servicio en el host'''


        def __init__(self, attrs):
                try:
                        self.name = attrs['name']


                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing
<SERVICE> tag: %s\n\n" % str(e)
                sys.exit(1)


                self.parameters= {}
                for k, v in attrs.iteritems():
                        if (k != 'name'):
                                self.parameters[k]=v


        def check_exists(self, templates_file, template_name):
                try:
                        doc = etree.parse (templates_file)      #Abrimos archivo de plantillas


                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing XML
file: %s\n\n" % str(e)
```

```
                sys.exit(1)

                templates = doc.findall('vm')          #Buscamos todas las VMs definidas

                exist = 0       #Existe el servicio
                params = 0      #Existen los parametros
                for vm in templates:
                        if (vm.attrib['name'] == template_name):       #VM encontrada
                                services = vm.findall('service')
                                for service in services:
                                        if (service.attrib['name'] == self.name):
                                                exist=1                                 #Servicio
encontrado
                                                self.init = service.attrib['init']     #Guardamos
comando inicio
                                                break
                                if exist:
                                        paramet = service.findall('param')

                                        if(len(paramet) != len(self.parameters)):     #No coincide
el numero de parametros
                                                break

                                        list_param = []
                                        value_param = []
                                        for param in paramet:
                                                list_param.append(param.attrib['name'])
        #Creamos una lista con los parametros definidos en la plantilla
                                                value_param.append((param.attrib['name'],
param.attrib['mod']))


        if(len(set(list_param).intersection(self.parameters.keys())) == len(paramet)):
                                                params = 1              #Si la interseccion de las
2 listas coincide con el numero de parametros, correcto

                                        for param in value_param:

                                                self.init += ' %s %s ' % (param[1],
self.parameters.get(param[0]))

                                        break

                return (exist, params)          #devolvemos resultado de la busqueda

        def start_service(self, fd, vnc_sendkeys, host_ip, vnc_display):
                try:
```

```
                        script_line = 'echo -e \"send_string %s\\nsend_key Return\\nq\\n\" |
python %s %s %i\n'\
                                        % (self.init, vnc_sendkeys, host_ip, vnc_display + 5900)
                        fd.write(script_line)
                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s
file" % fd


        def print_service(self):
                print "\t\tSERVICE:\n\t\t", self.name


        #def __del__(self):


class Event:
        '''Clase que implementa un evento a lanzar en el host'''


        def __init__(self, attrs):
                try:
                        self.at = attrs['at']


                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing <EVENT>
tag: %s\n\n" % str(e)
                sys.exit(1)


                self.attacks = []


        def add_attack(self, attack):
                self.attacks.append(attack)


        def get_attacks(self):
                return self.attacks


        def create_event(self, fd, vnc_sendkeys, host_ip, vnc_display):
                for attack in self.attacks:
                        attack.run_attack(fd, vnc_sendkeys, host_ip, vnc_display, self.at)


        def print_event(self):
                print "\t\tEVENT:\n\t\t", self.at
                for attack in self.attacks:
                        attack.print_attack()


        #def __del__(self):


class Attack:
```

```
        '''Clase que implementa un ataque a lanzar en el host'''

        def __init__(self, attrs):
                try:
                        self.template = attrs['template']

                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing <ATTACK>
tag: %s\n\n" % str(e)
                sys.exit(1)

                self.parameters= {}
                for k, v in attrs.iteritems():
                        if (k != 'template'):
                                self.parameters[k]=v

        def filter_string(self, string):
                pattern = '\s+' #Patron a buscar
                p = re.compile(pattern)
                return p.sub(' ', string)       #Sustituimos varios espacios en blanco por uno
solo

        def check_exists(self, templates_file, template_name):
                try:
                        doc = etree.parse (templates_file)      #Abrimos archivo de plantillas

                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing XML
file: %s\n\n" % str(e)
                sys.exit(1)

                templates = doc.findall('vm')           #Buscamos todas las VMs definidas

                exist=0         #Existe el ataque indicado
                typ=0           #Existe el tipo de ataque
                params=0        #Existen los parametros del ataque

                for vm in templates:
                        if (vm.attrib['name'] == template_name):        #VM encontrada
                                attacks = vm.findall('attack')          #Buscamos todos los
ataques definidos
                                for attack in attacks:
                                        if (attack.attrib['name'] == self.template):
                                                exist=1                 #Ataque encontrado
                                                self.attack_command = attack.attrib['command']
```

```
                                        break
                                if exist:
                                        types = attack.findall('type') #Buscamos todos los tipos
definidos

                                        if ('type' in self.parameters): #Se ha definido un tipo de
ataque en el escenario
                                                for typee in types:
                                                        if (typee.attrib['name'] ==
self.parameters['type']):
                                                                typ=1   #Tipo encontrado
                                                                self.attack_command += ' %s' %
typee.attrib['parameter']
                                                                break
                                                if typ:
                                                        paramet = typee.findall('param')

                                                        if(len(paramet) != len(self.parameters)-
1):     #No coincide el numero de parametros
                                                                break

                                                        list_param = []
                                                        value_param = []
                                                        for param in paramet:

        list_param.append(param.attrib['name'])             #Creamos una lista con los
parametros definidos en la plantilla

        value_param.append((param.attrib['name'], param.attrib['mod']))


        if(len(set(list_param).intersection(self.parameters.keys())) == len(paramet)):
                                                                params = 1              #Si la
interseccion de las 2 listas coincide con el numero de parametros, correcto

                                                        for param in value_param:

                                                                self.attack_command += ' %s %s' %
(param[1], self.parameters.get(param[0]))

                                                        break
                                        else:
                                                if(len(types)==0):     #Si no es necesario
definir el tipo de ataque
                                                        typ=1

                                                        params=1

                                                break
```

```
                return (exist, typ, params)            #devolvemos resultado de la busqueda


        def run_attack(self, fd, vnc_sendkeys, host_ip, vnc_display, time):
                try:
                        script_line = 'echo -e \"send_string echo %s | at now + %s\\nsend_key
Return\\nq\\n\" | python %s %s %i\n'\
                                        % (self.filter_string(self.attack_command), time,
vnc_sendkeys, host_ip, vnc_display + 5900)
                        fd.write(script_line)
                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s
file" % fd


        def print_attack(self):
                print "\t\t\tATTACK:\n\t\t\t", self.template


        #def __del__(self):


class Ipchain:
        '''Clase que implementa una regla de ipchain'''


        def __init__(self, attrs):
                try:
                        self.rule = attrs['rule']


                except Exception, e:
                print "\n\n\t[+] An unexpected exception was encountered while parsing
<IPCHAINS> tag: %s\n\n" % str(e)
                sys.exit(1)


        def create_rule(self, fd, vnc_sendkeys, host_ip, vnc_display):
                try:
                        script_line = 'echo -e \"send_string iptables %s\\nsend_key
Return\\nq\\n\" | python %s %s %i\n'\
                                        % (self.rule, vnc_sendkeys, host_ip, vnc_display + 5900)
                        fd.write(script_line)
                except IOError:
                print "\n\n\t[+] An error was encountered while writing output script in %s
file" % fd


        def print_ipchain(self):
                print "\t\tIPCHAIN:\n\t\t", self.rule


        #def __del__(self):
```

```
def main():


        parser = OptionParser(version="%prog 1.0")

        parser.add_option("-f", "--file", action="store", type="string", dest="filename",
help="CONFIGURATION file [default: %default]", default="./config.cfg")


        (options, args) = parser.parse_args()


        if (len(args) != 0):

        parser.error("wrong number of arguments")


        # Chequemos si existe el archivo de configuracion

        try:

                open(options.filename)


                try:

                        config_file = ConfigParser.SafeConfigParser()         #Inicializamos el
ConfigParser

                        config_file.read(options.filename)                    #Leemos el archivo
de configuracion


                        vnc_sendkeys = config_file.get('main', 'vnc_sendkeys')

                        input_stage = config_file.get('main', 'input_stage')

                        templates = config_file.get('main', 'templates')

                        output_script = config_file.get('main', 'output_script')



                except Exception, e:

                print "\n\n\t[+] An unexpected exception was encountered while parsing CONFIG
file. %s\n\n" % str(e)

                sys.exit(1)


        except IOError, e:

                print "\n\n\t[+] An unexpected exception was encountered while parsing CONFIG
file. %s\n\n" % str(e)

                sys.exit(1)



        template_parser = Template_parser(templates)

        template_parser.parse_template()

        parser = Stage_parser(input_stage, templates)

        stage = parser.parse_xml()

        #stage.print_stage()

        stage.configure_stage(output_script, vnc_sendkeys)

        print "\n\n[+] Stage parsed correctly!"

        print "\n[+] Output script is on:", output_script, "\n\n"
```

```
if __name__ == '__main__':
  main()
```

Archivo 'vnc_send_keys.*py*'

```
import socket
import time
import struct
import sys

class RFBClient:

        keycodes = {
                'BackSpace' : 0xff08,
                'Tab' :   0xff09,
                'Return' :  0xff0d,
                'Escape' :  0xff1b,
                'Insert' :  0xff63,
                'Delete' :  0xffff,
                'Home' :   0xff50,
                'End' :   0xff57,
                'PageUp' :  0xff55,
                'PageDown' : 0xff56,
                'Left' :   0xff51,
                'Up' :    0xff52,
                'Right' :  0xff53,
                'Down' :   0xff54,
                'F1' :    0xffbe,
                'F2' :    0xffbf,
                'F3' :    0xffc0,
                'F4' :    0xffc1,
                'F5' :    0xffc2,
                'F6' :    0xffc3,
                'F7' :    0xffc4,
                'F8' :    0xffc5,
                'F9' :    0xffc6,
                'F10' :   0xffc7,
                'F11' :   0xffc8,
                'F12' :   0xffc9,
                'F13' :   0xFFCA,
                'F14' :   0xFFCB,
                'F15' :   0xFFCC,
                'F16' :   0xFFCD,
                'F17' :   0xFFCE,
                'F18' :   0xFFCF,
                'F19' :   0xFFD0,
                'F20' :   0xFFD1,
                'ShiftLeft' : 0xffe1,
                'ShiftRight' : 0xffe2,
                'ControlLeft' : 0xffe3,
                'ControlRight' : 0xffe4,
                'MetaLeft' : 0xffe7,
                'MetaRight' : 0xffe8,
                'AltLeft' :  0xffe9,
                'AltRight' : 0xffea,

                'Scroll_Lock' : 0xFF14,
                'Sys_Req' :  0xFF15,
                'Num_Lock' : 0xFF7F,
                'Caps_Lock' : 0xFFE5,
                'Pause' :  0xFF13,
                'Super_L' :  0xFFEB,
                'Super_R' :  0xFFEC,
                'Hyper_L' :  0xFFED,
```

```
        'Hyper_R' : 0xFFEE,

        'KP_0' :   0xFFB0,
        'KP_1' :   0xFFB1,
        'KP_2' :   0xFFB2,
        'KP_3' :   0xFFB3,
        'KP_4' :   0xFFB4,
        'KP_5' :   0xFFB5,
        'KP_6' :   0xFFB6,
        'KP_7' :   0xFFB7,
        'KP_8' :   0xFFB8,
        'KP_9' :   0xFFB9,
        'KP_Enter' : 0xFF8D,

    'Equal' : 0xFFFA,
        'Underline' : 0xFFFB,
}

def __init__(self, address):
        # conectamos
        self.s = s = socket.socket()
        s.connect(address)

        # negociamos version
        version = s.recv(12)
        print version.strip()
        s.sendall(version)

        # negociamos seguridad
        security_types = ord(s.recv(1))
        for i in range(security_types):
                s.recv(1)

        s.sendall(chr(1))
        struct.unpack('!I', s.recv(4))

        # iniciamos cliente (compartido)
        s.sendall(chr(1))

def __send_keycode(self, keycode):
        if (keycode==0xFFFB):
                self.s.sendall(struct.pack("!BBxxI", 4, 1, 0xffe1))
                time.sleep(0.05)
                self.s.sendall(struct.pack("!BBxxI", 4, 1, ord('\'')))
                time.sleep(0.05)
                self.s.sendall(struct.pack("!BBxxI", 4, 0, 0xffe1))
                time.sleep(0.05)
                self.s.sendall(struct.pack("!BBxxI", 4, 0, ord('\'')))
                self.s.sendall(struct.pack("!BBxxI", 4, 1, 0xffe1))
                time.sleep(0.05)
                self.s.sendall(struct.pack("!BBxxI", 4, 0, 0xffe1))
                time.sleep(0.05)

        else:
                self.s.sendall(struct.pack("!BBxxI", 4, 1, keycode))
                time.sleep(0.05)
                self.s.sendall(struct.pack("!BBxxI", 4, 0, keycode))


def send_string(self, string):
        for c in string:
                self.__send_keycode(ord(c))

def send_key(self, key):
        self.__send_keycode(self.keycodes[key])

def parse_command(self, line):
        parsed=line.partition(' ')

        if(parsed[0]=="send_string"):
```

```
                        client.send_string(parsed[2])
                elif(parsed[0]=="send_key"):
                        client.send_key(parsed[2].replace('\n',''));
                elif(parsed[0]=="sleep"):
                        time.sleep(int(parsed[2]))
                else:
                        print "\n\t[!] Comando invalido\n"


if(len(sys.argv) == 4):
        #abrimos archivo de ordenes
        fd = open(sys.argv[3], "r")

        #conectamos con los datos suministrados
        client = RFBClient((sys.argv[1], int(sys.argv[2])))

        #leemos linea por linea y parseamos el tipo de orden
        fileList = fd.readlines()
        for fileLine in fileList:
                client.parse_command(fileLine)

        #cerramos el descriptor
        fd.close()

elif(len(sys.argv) == 3):
        #conectamos con los datos suministrados
        client = RFBClient((sys.argv[1], int(sys.argv[2])))

        line = sys.stdin.readline()

        while (line!="q\n"):
                client.parse_command(line)
                line = sys.stdin.readline()

else:
        print "\n\t[+] Uso: " + sys.argv[0] + " servidor_VNC puerto archivo_ordenes\n"
```

ANEXO B – Instalación de Ubuntu

Como ya se indicó en el capítulo 3, el sistema operativo elegido para instalarlo en los diferentes equipos que conforman la red del laboratorio, es Ubuntu en su versión 10.04. El primer paso será descargar la *ISO* de la página web oficial[28], asegurándonos de que se elige la arquitectura de procesador acorde a nuestros equipos, en este caso 32 *bits*. Podemos ver esto en la *ilustración 105*.

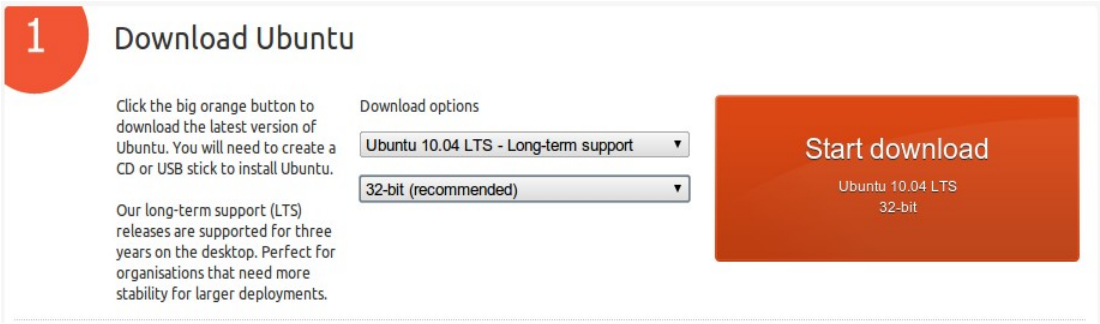

Ilustración 105: Descarga de Ubuntu 10.04.

Una vez que hemos descargado la ISO, el siguiente paso será guardarla en algún dispositivo externo que podamos conectar a las máquinas del laboratorio y proceder a su instalación, *i.e.: cd-rom*, *dvd*, *pendrive*, etc. En nuestro caso se ha optado por hacerlo en un *pendrive*, por lo que necesitamos un programa que sea capaz de manipular y guardar adecuadamente la imagen del sistema operativo en un *pendrive*, en nuestro caso *UNetbootin*[29].

Después de su descarga para la plataforma en la que estemos trabajando, debemos de introducir el *pendrive* usado para la instalación en las diversas máquinas en el ordenador, y posteriormente correr el programa con permisos de administrador. En la pantalla principal debemos seleccionar la opción "*DiscoImagen – ISO"*, y la ruta donde hemos descargado la imagen del sistema operativo. A continuación seleccionaremos la unidad *usb* correspondiente a nuestro *pendrive*, y finalmente le daremos al botón aceptar.
Podemos observar lo explicado a continuación en la *ilustración 106*.

28 Ubuntu 10.04 (2011), http://www.ubuntu.com/download/ubuntu/download

29 Unetbootin (2011), http://unetbootin.sourceforge.net/

Ilustración 106: Configuración de UNetbootin.

Con esto, ya tendremos el *pendrive* preparado con Ubuntu 10.04, y listo para instalarlo en los equipos del laboratorio.

Para poder llevar a cabo la instalación, se debe consultar la documentación de los equipos del laboratorio, en donde se explicará como configurar adecuadamente la BIOS de éstos, de forma que permitan el arranque desde un *pendrive*. Una vez hecho esto, arrancaremos, y nos aparecerá el menú de UNetbootin, donde debemos selecionar la opción "*Default*", como observamos en la *ilustración 107*.

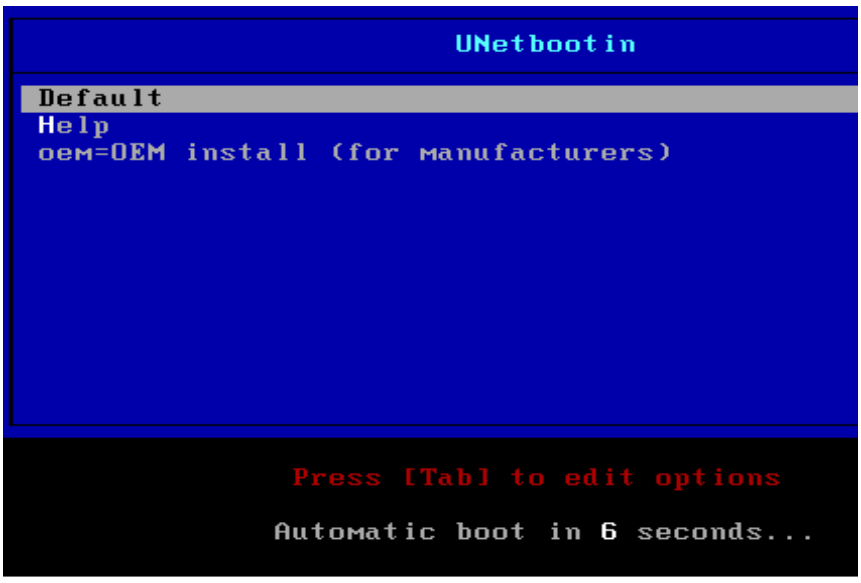

Ilustración 107: Menú de Unetbootin.

Después de realizar el proceso de carga y arranque de Ubuntu, nos aparecerá la pantalla principal, en la que se nos insta a probarlo o instalarlo, en nuestra caso optaremos por la segunda opción. En la *ilustración 108* podemos ver esto.

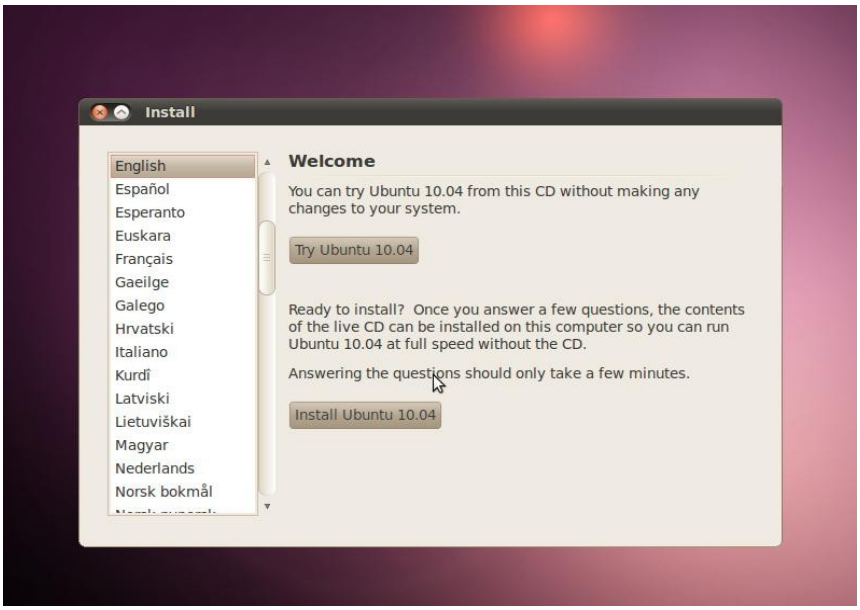

Ilustración 108: Pantalla principal de la instalación de Ubuntu.

Después de seleccionar el idioma deseado, en nuestro caso Español, nos aparecerá la pantalla de selección de ubicación y huso horario, como observamos en la *ilustración 109*. En nuestro caso seleccionamos Madrid.

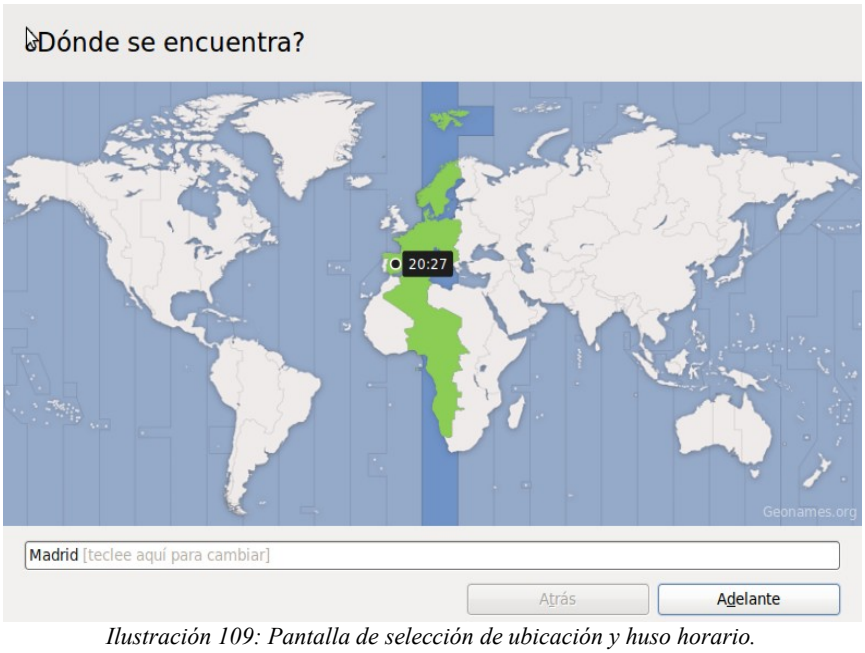

Ilustración 109: Pantalla de selección de ubicación y huso horario.

A continuación, como podemos ver en la *ilustración 110*, se nos presentará la opción de selección de teclado. En nuestro caso seleccionamos España.

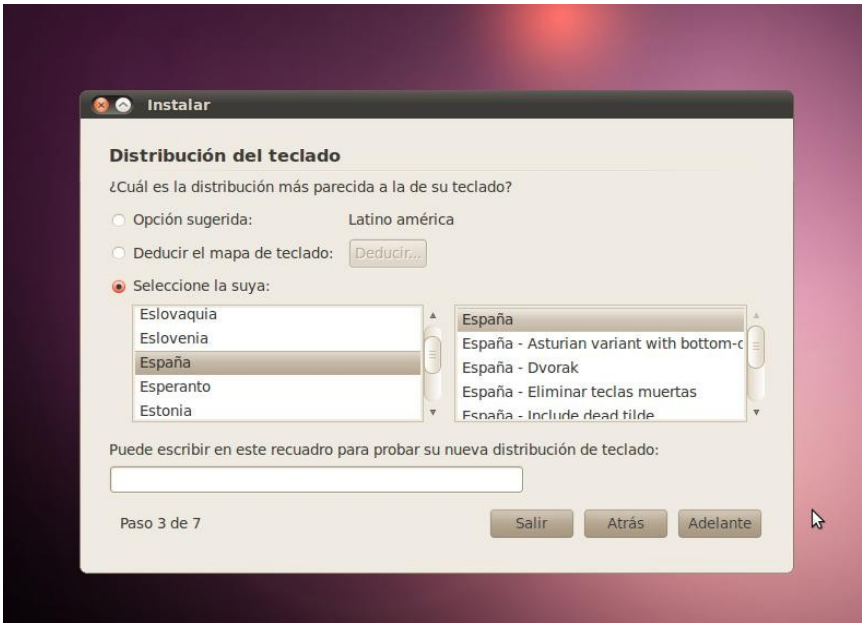

Ilustración 110: Pantalla de selección de distribución de teclado.

Una vez hecho el paso anterior, nos encontramos con la fase de particionado del disco duro y selección de particiones. Tenemos 3 opciones:

- Usar todo el disco duro
- Usar el disco entero particionándolo manualmente
- Compartir disco duro con otros sistemas operativos (en caso de que lo detectase).

Como en nuestro caso usaremos los equipos completamente para el proyecto, elegiremos la primera opción, como se observa en la *ilustración 111*.

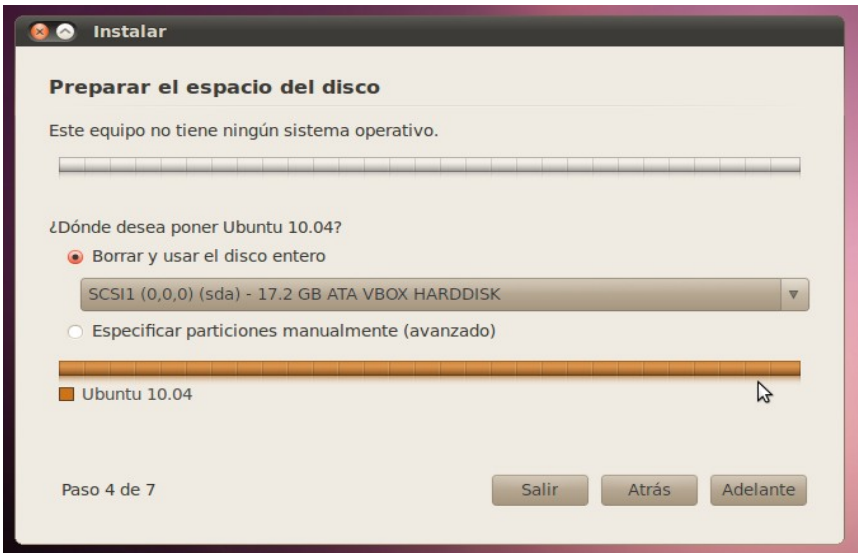

Ilustración 111: Pantalla de particionado.

Una vez realizado el paso anterior, deberemos de introducir una serie de parámetros para

acceder a la máquina, como observamos en la *ilustración 112*.

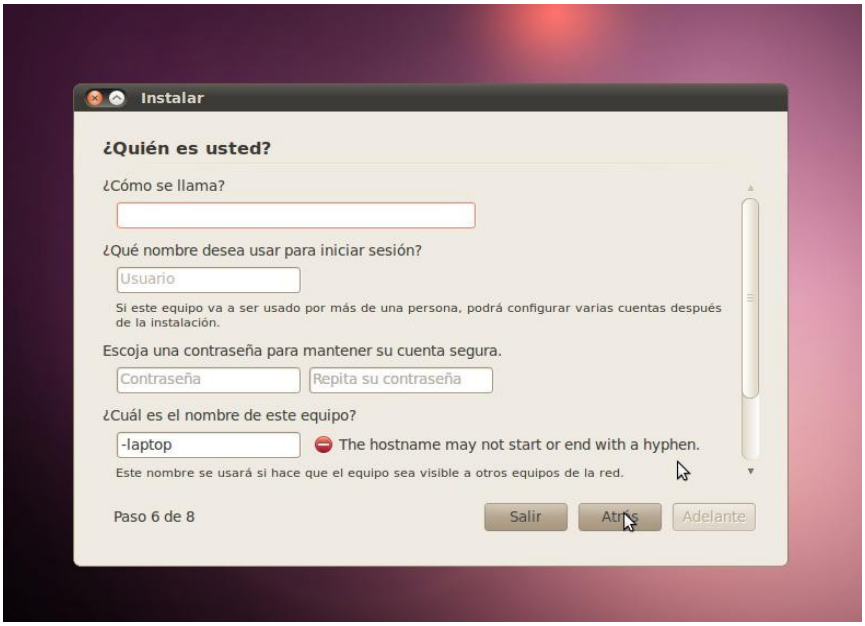

Ilustración 112: Pantalla de configuración de usuario y nombre del equipo.

Figura 5.8.

Con esto estaría finalizada la configuración, por lo que empezaría la instalación automática de todos los componentes del sistema operativo.

ANEXO C – Contenido del CD-ROM

En el CD-ROM adjunto a la memoria del Trabajo de Fin de Carrera se han incluido los siguientes elementos:

- *pfc.py*: Código fuente comentado de la aplicación principal desarrollada en el proyecto.

- *vnc_send_keys.py*: Código fuente del *script* desarrollado para poder comunicarnos con las máquinas virtuales a través del protocolo VNC.

- d*emo.ogv*: Video con una demostración del funcionamiento del proyecto.